

Java™ magazine

By and for the Java community



GROWING ON OPEN

Java provides
open source
foundation for
**AgroSense and
Hadoop**

14

THE ECLIPSE
FOUNDATION'S
MIKE MILINKOVICH

38

JAVAFX IN
SPRING

//table of contents /

COMMUNITY

02

From the Editor

04

Java Nation

News from JavaOne, plus people, events, and books

JAVA TECH

30

New to Java

Can You Teach Testing to Beginners?

Michael Kölling wraps up his series on BlueJ with a look at its interactive-testing support.

34

Java Architect

Exploring Lambda Expressions for the Java Language and the JVM

Ben Evans, Martijn Verburg, and Trisha Gee help you get ready for lambda expressions in Java SE 8.

42

Mobile and Embedded

Wirelessly Recover Your Device's Address Book

Vikram Goyal explains how to add a recovery model to the backup capability of your device's address book.

46

Mobile and Embedded

Get Started with Java SE for Embedded Devices on Raspberry Pi

Bill Courington and Gary Collins walk you through getting Linux and Java SE for Embedded Devices to run on the Raspberry Pi in less than an hour.

55

Polyglot Programmer

Building Actor-Based Systems Using the Akka Framework

Ted Neward shows you how to use Akka's open source actor model implementation to build distributed systems.

60

Fix This

Take our JDBC code challenge.



14

JCP Executive Series

Q&A WITH MIKE MILINKOVICH

The Eclipse Foundation's executive director assesses the state of Java and the JCP.

20

Java in Action

GROWING ON OPEN

AgroSense provides an all-Java open source platform for sustainable farming and precision agriculture.

25

Java in Action

AN ENGINE FOR BIG DATA

Hadoop uses Java for large-scale analytics.

38

Rich Client

JAVAFX IN SPRING

Stephen Chin shows you why to use the Spring framework on the client.

//from the editor /

T

hey say that it takes a village to raise a child. As a parent, I wholeheartedly agree. The teachers, coaches, friends, and family who regularly interact with my son are influencing the person he is becoming—and giving him access to a much greater range of specialties and world views than his father and I could provide on our own. Open source software is not so different. It, too, is raised by a global village of contributors, each of whom offers his or her own special breed of knowledge. This collaborative model keeps the software evolving and improving and encourages innovation.

In this issue, we explore how Java provides a platform for two organizations to build their own open source projects. 2012 Duke's Choice Award winner the [AgroSense Project](#) built an open source farm management system on Java and the NetBeans Platform that gives farmers the information they need to manage their crops. Another 2012 Duke's Choice Award winner, [Hadoop](#), is an open source software platform written in Java that enables businesses to unlock potential value from big data. Because Hadoop is open source, decisions about it are made by consensus. "You can't just do things unilaterally," Hadoop creator Doug Cutting tells us.

We also get an open source perspective from Eclipse Foundation Executive Director [Mike Milinkovich](#) in our Java Community Process Executive Committee member interview. Plus: Simon Phipps of the Open Source Initiative tells us what's driving open source today in a [video interview](#) with Tori Wieldt, and [Ted Neward](#) shows us how to use Akka, an open source actor model implementation.

It's been a great year at *Java Magazine*. Looking ahead to 2013, we are open to your feedback so that we, too, can keep growing and innovating.

Caroline Kvitka, Editor in Chief [BIO](#)

PHOTOGRAPH BY BOB ADLER



FIND YOUR JUG HERE

One of the most elevating things in the world is to build up a community where you can hang out with your geek friends, educate each other, create values, and give experience to you members.

Csaba Toth
Nashville, TN Java Users' Group (NJUG)

[LEARN MORE](#)



ORACLE®

//send us your feedback /
We'll review all suggestions for future improvements. Depending on volume, some messages may not get a direct reply.





Answers. Fast.

Subscribe and gain instant cutting-edge development insight from the world's most trusted tech publishers — all in one on-demand digital library.

- Get unlimited access to 23,000+ online books and training videos from 100+ publishers
- Search full text to efficiently get the answers you need
- Make notes and organize content into folders you control
- Elevate your career with thousands of top business and professional development titles



ANYTIME, ANYWHERE:     

Access on your desktop, notebook, tablet or mobile device.

Start a free unlimited-access trial today safaribooksonline.com/javamag

MAKE THE FUTURE JAVA

The theme of JavaOne 2012, held September 30–October 4, was “Make the Future Java.” Throughout the week, attendees explored Java’s continued role as the most popular, complete, productive, secure, and innovative development platform as well as its open, transparent, collaborative, and community-driven evolution. At Sunday’s Strategy keynote, Oracle’s **Hasan Rizvi** detailed the three factors critical to Java’s success: technology innovation, community participation, and Oracle’s leadership/stewardship. Under technology, he noted Macintosh OS X and Linux ARM support on Java SE, open sourcing of JavaFX by the end of 2012, the release of Oracle Java Embedded Suite 7.0, and multiple releases on the Java EE side. Under community, he said that the Java Community Process (JCP) continues, with new JSR activity and Java user group participation up 25 percent since last year.

PHOTOGRAPHS BY HARTMANN STUDIOS



Brian Goetz

Mark Reinhold



Georges Saab



Nandini Ramani

Under stewardship, he noted Oracle’s continued outreach—with four regional JavaOne conferences last year and the launch of *Java Magazine*. Here are highlights from Sunday’s Strategy and Technical keynotes:

Java 8/Java 9

Oracle’s **Georges Saab** discussed the upcoming JDK 8 release—including Project Lambda and Project Nashorn (a modern implementation of JavaScript running on the Java Virtual Machine). He noted that Nashorn functionality has already been used internally in NetBeans 7.3, and

announced that Oracle plans to contribute the implementation to OpenJDK.

Oracle’s **Brian Goetz** explored language and library features planned for Java SE 8, including lambda expressions and better parallel libraries. These feature changes both simplify code and free up libraries to more effectively use parallelism.

Oracle’s **Mark Reinhold** urged developers to get involved in the Java 8 development process—getting the weekly builds, trying out their current code, and trying out the new features.

//java nation / JavaOne 2012 /



Left: Nike's Nicole Otto presented Oracle's Cameron Purdy with a NikeFuel accelerometer wrist band so that he can track his activity.

Right: Dr. Robert Ballard, oceanographer and National Geographic explorer in residence, presented an overview of the cutting-edge technology used in deep-sea explorations.

Saab also explored Java SE 9 and beyond—Jigsaw modularity; Project Penrose for interoperability with OSGi; improved multitenancy for Java in the cloud; and Project Sumatra, an OpenJDK project targeted at bringing Java to heterogeneous platforms.

JavaFX

Oracle's **Nandini Ramani** announced that a developer preview of JavaFX on Linux ARM as well as JavaFX Scene Builder on Linux are available for download. She noted other JavaFX 2 milestones including releases on Microsoft Windows, Macintosh OS X, and Linux; the JavaFX Scene Builder tool; the JavaFX WebView component in NetBeans 7.3; and an OpenJFX project in OpenJDK. In JDK 8, JavaFX will offer 3-D and third-party component integration, she said.

Embedded

Ramani discussed the latest on Java in the embedded space, declaring this to be "the next IT revolution," with Java as the ideal technology for the ecosystem. (See "[A Huge Opportunity](#)

[in Small Things](#)" for details of Oracle's product announcements.) Ramani encouraged developers to download the latest releases of Oracle Java SE Embedded and try them out.

Java EE 7/Java EE 8

Oracle's **Cameron Purdy** summarized the latest developments in the enterprise space: greater developer productivity in Java EE 6 and portability between platforms, vendors, and even cloud to cloud. The earliest version of the Java EE 7 Software Development Kit (SDK) is now available for download—in GlassFish 4—with WebSocket support, better JSON support, and more. The final release is scheduled for April 2013.

Looking forward to Java EE 8, Oracle's **Arun Gupta** explored how the platform will provide multitenancy for applications, modularity based on Jigsaw, and cloud architecture. He also talked about Project Avatar, for designing an end-to-end framework for building HTML5 applications, and Project Easel, an advanced tooling capability in NetBeans for HTML5.

JavaOne Content on Video

JavaOne content is available on video in three categories: [keynote highlights](#); [full keynotes](#); and all [sessions](#), [birds-of-a-feather \(BOF\)](#) sessions, and [tutorials](#).



Welcome to JavaOne 2012

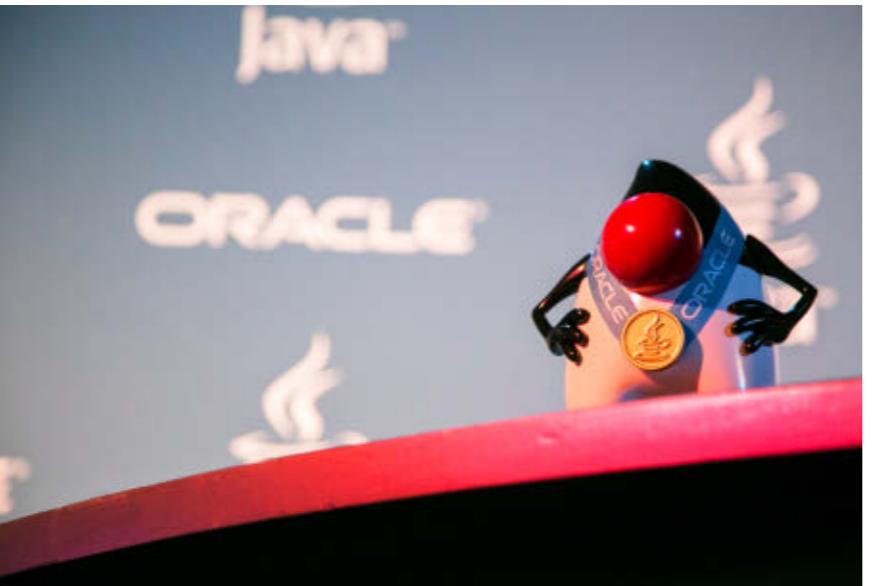


Strategy keynote highlights



Technical keynote highlights

2012 DUKE'S CHOICE AWARDEES RECOGNIZED



Left: The 2012 Duke's Choice Award.

Right: Oracle's Orla Nichorcora, senior director business development, presents a 2012 Duke's Choice Award to the creators of UNHCR's Level One registration tool, Doudou Stanylas Matayo (left) and Abdouraouf Gnon-Konde.



The winners of the 2012 Duke's Choice Awards were recognized at a ceremony on the opening day of JavaOne. "This year's Duke's Choice Awards winners are spearheading a truly diverse and creative set of Java-based projects, and their efforts are invaluable to the Java community," says Oracle's **Peter Utzschneider**.

One of the winners, [The United Nations High Commissioner for Refugees \(UNHCR\)](#), was also recognized at Oracle Day in Kenya on October 12 and presented with its Duke's Choice Award. To help facilitate its mission of humanitarian relief, UNHCR developed a light-client Java application on the NetBeans platform that collects information on the number of refugees and their water, food, housing, health, and other needs in the field and combines that with geocoding information from various sources.

"UNHCR, along with all of this year's recipients, demonstrates the remarkable work being done by Java community members across the world," says Utzschneider.

PHOTOGRAPHS BY ORANGE PHOTOGRAPHY,
HAIG ANYONYI

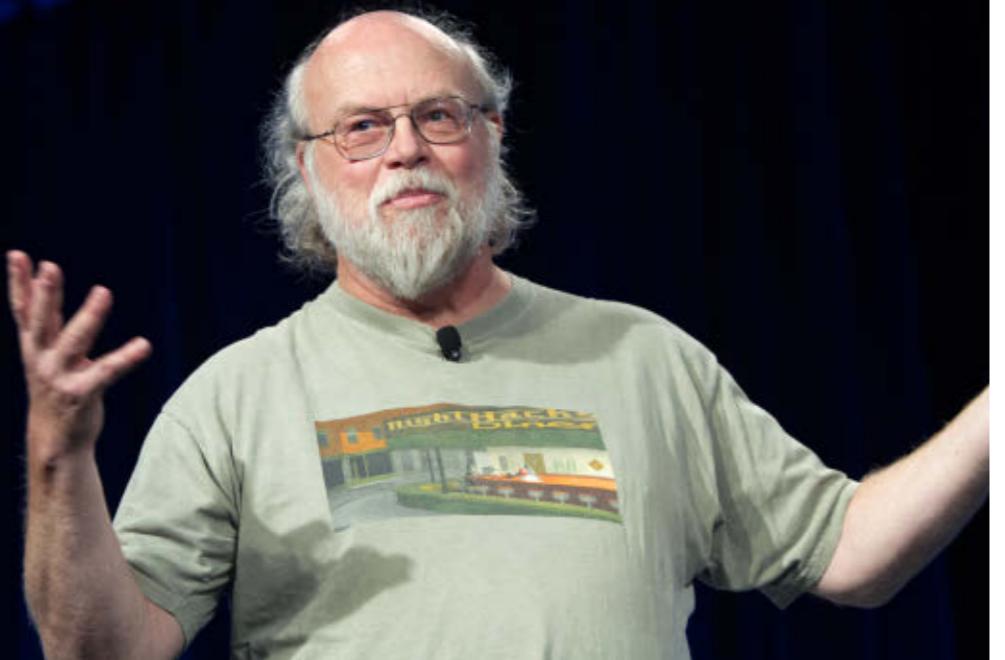
Project Sumatra Update

[Project Sumatra](#) is a new OpenJDK project that will enable Java applications to take advantage of graphical processing units (GPUs) to improve application performance. The Sumatra effort is led by Oracle's John Coomes, who has worked on the Java HotSpot VM for almost a decade. Initial development will focus on code generation, garbage collection, and runtime development for offloading processing to GPUs. Subsequently, [Java 8 lambda expressions](#) will be utilized to further increase application parallelism.

At the JavaOne Community keynote, Gary Frost of AMD (which has joined Project Sumatra) talked about the Sumatra effort and demonstrated GPU speedups attained using the related [Aparapi](#) (A Parallel API) open source project. Aparapi executes data-parallel code on GPUs by converting Java bytecode to [OpenCL](#), which runs on a wide variety of GPUs. In applications that plotted Mandelbrot fractals, played the [Game of Life](#), and solved N-body gravitational physics equations for 10,000 celestial bodies, Frost's demonstrations showed speedups of up to 10 times using Aparapi compared with traditional Java threads processing.

As with all OpenJDK projects, the community is invited to participate in Project Sumatra. While the project is still just getting started, ultimately developers will be needed to test releases, suggest improvements, fix bugs, and more.

//java nation / JavaOne 2012 /



Clockwise from top: James Gosling describes the Wave Glider; Paul Perrone plays with robots; Sharat Chander proudly displays his JavaOne badge.

Thursday's Community keynote reinforced the idea that JavaOne is by and for the community.

Sharat Chander, JavaOne community chairperson, noted that 60 percent of the material at the 2012 JavaOne conference was presented by Java community members, and he encouraged even higher participation next year.

Oracle's **Donald Smith** explored the importance of community in terms of fostering innovation.

PHOTOGRAPHS BY HARTMANN STUDIOS

BY AND FOR THE COMMUNITY

ion. He invited panelists from Cloudera, Eclipse, Eucalyptus, Perrone Robotics, and Twitter onstage to further discuss the idea of innovation. Panelist **Mike Milinkovich**, executive director of the Eclipse Foundation, noted, “The more open you make your innovation process, the more ideas are challenged, and the more developers are focused on justifying their choices all the way through the process.”

Continuing the topic of innovation, Oracle's

better open standards. "Come join us, and make your ecosystem better!" urged Verburg.

Next, **Paul Perrone** of Perrone Robotics returned to profile the latest in his company's robotics work around Java—including the AARDBOTS family of smaller robotic vehicles, running the Perrone MAX platform on top of the Java Virtual Machine (JVM). Perrone took his "Rumbles" four-wheeled robot out for a spin—a roaming, ARM-based security-bot vehicle, complete with ultrasonic and "cliff" sensors.

Then, a mysterious voice from offstage pronounced, "I've got some toys"—proving to be surprise guest **James Gosling**, there to explore his cutting-edge work with Liquid Robotics. Gosling demonstrated real-time satellite tracking of several Wave Gliders currently at sea, noting that Java is actually particularly good at artificial intelligence applications—due to the language having garbage collection, which facilitates complex data structures.

Finally, Chander again took the stage, where he passed the JavaOne Community Chairperson baton to Oracle Java Technology Evangelist **Stephen Chin**. Wearing full motorcycle gear, Chin noted that he'll soon be touring Europe by motorcycle on his [NightHacking Tour](#), meeting Java community members and streaming live via Ustream. Watch for coverage in our next issue.

GEEK BIKE RIDE

Following the tradition of [JavaOne Latin America 2011](#), a Geek Bike Ride marked the beginning of JavaOne 2012 on a gorgeous day in San Francisco. About 50 Java developers gathered at Fisherman's Wharf, rode or skated through Fort Mason and Crissy Field, crossed the Golden Gate Bridge, and finished in Sausalito. Riders returned to San Francisco by ferry after a well-earned lunch. Participants donned Duke bike jerseys, sponsored by Oracle Technology Network. To receive a jersey, participants had to answer a question about Java. Questions included "Who is the father of Java?" "What's the biggest Java conference in San Francisco?" and "Name one Duke's Choice Award winner from this year," to which participant **Rérgina ten Bruggencate** answered, "Me!"



Clockwise from top: Geek Bike Ride participants paused for a photo op at the water's edge; SouJava's Juggy joined in on the ride; bikers enjoyed a leisurely pace.

PHOTOGRAPHS BY YOSHIO TERADA

A HUGE OPPORTUNITY IN SMALL THINGS

Addressing the strong demand for Java in the embedded market, Oracle hosted a new [Java Embedded @ JavaOne](#) event in San Francisco October 3–4. In his keynote address, Oracle's **Judson Althoff** said that devices are all around us, on 24/7, and connected all the time. The explosion of devices is the next IT revolution and Java is the right solution for this space, he said. He noted that Java embedded solutions provide a framework to provision, manage, and secure devices as well as the ability to aggregate, process, and analyze a multitude of data. Finally, he said, Java is one platform to program them all.

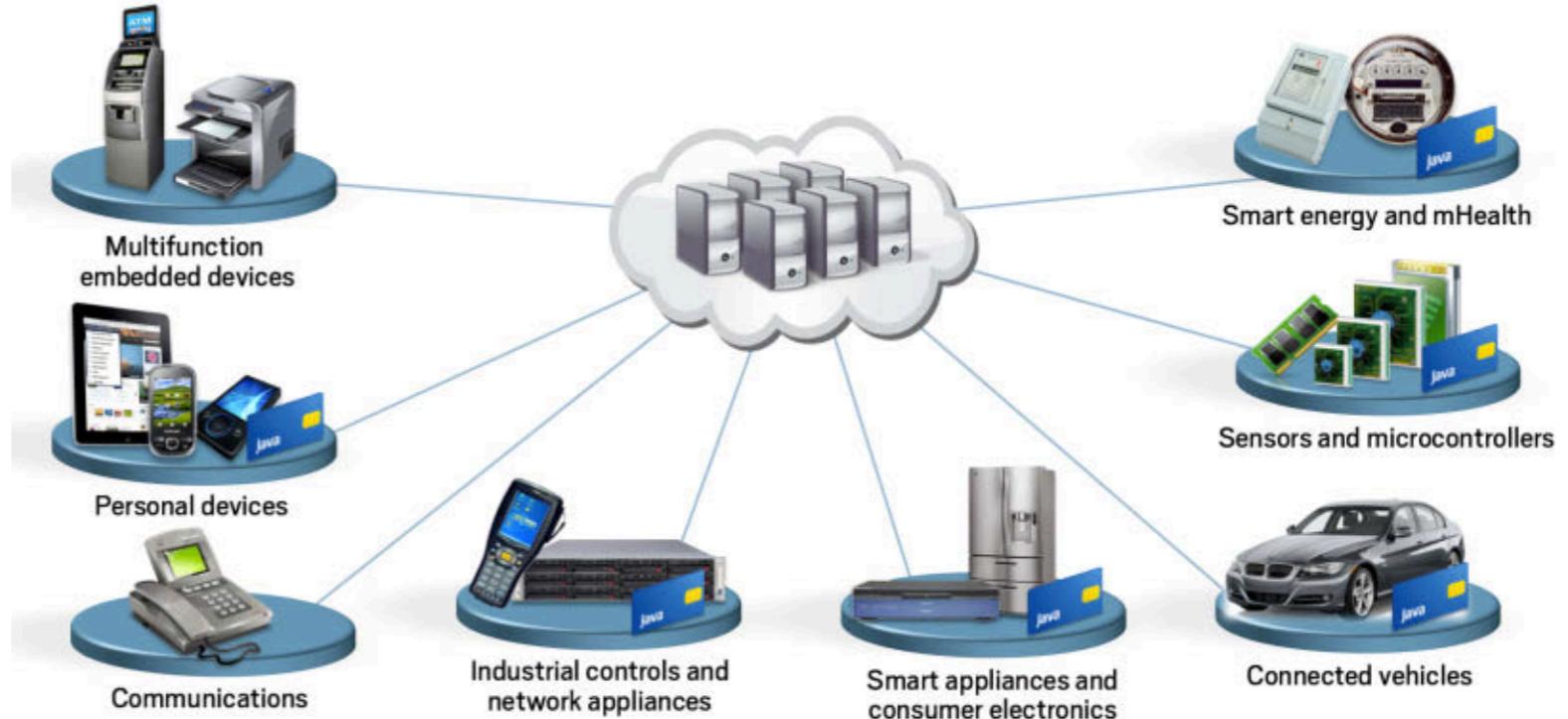
Terrance Barr, Java evangelist and Java ME expert, is enthusiastic about the huge opportunity. "It's the right time and right place for Java embedded," he said. "Oracle is looking for partners who want to take advantage of this next wave in IT."

The embedded space continues to heat up. At JavaOne, Cinterion launched [EHS5](#), an ultra-compact, high-speed machine-to-machine communication module providing secure wireless connectivity for a wide variety of industrial applications.

Oracle Unveils Embedded Products

Just before JavaOne, Oracle announced several products in the embedded space.

[Oracle Java ME Embedded 3.2](#) is a complete client Java runtime optimized for resource-



constrained, connected, embedded systems. It is designed and optimized to meet the unique requirements of small, embedded, low-power devices such as microcontrollers and other resource-constrained hardware without screens or user interfaces.

[Oracle Java Wireless Client 3.2](#) is built around an optimized Java ME implementation that delivers a feature-rich application environment for mass-market mobile devices.

[Java ME Software Development Kit \(SDK\) 3.2](#) provides a complete development environment for both Oracle Java ME Embedded 3.2 and

Oracle Java Wireless Client 3.2.

[Oracle Java Embedded Suite 7.0](#) is a packaged solution (based on Oracle Java SE Embedded 7, Java DB, and versions of GlassFish for Embedded Suite and Jersey Web Services Framework), created to provide value-added services for collecting, managing, and transmitting data to and from other embedded devices. It is a complete device-to-data-center solution subset for embedded systems.

WOMEN IN TECH

Every year, JavaOne hosts top-notch women technologists who are pursuing careers in IT. They are inspired technologists and community organizers.

This year, **Trisha Gee**, **Régina ten Bruggencate**, and **Saskia Vermeer-Ooms**—three seasoned developers with more than a decade of programming experience—talked about inspiring more women to be active in the community and to pursue careers in programming. They are leaders of [Duchess](#), a global network that connects women involved in Java. The network has 500 members in 60 countries and a strong Web presence. In her session, “The Problem with Women: A Technical Approach,” Gee recommended that women speak at and organize events, blog, and share their passion for programming—and not mention their gender.

Fabiane Nardon and **Yara Senger** are developers and active community leaders in Brazil. Nardon is a frequent speaker and a member of program committees at various Java conferences. Her JavaOne session covered auto-scaling Web-based Java applications. Senger, the president of the SouJava Java user group (JUG) and a cofounder of Globalcode, presented sessions on the future of Java and on Java APIs for electronic devices and external boards. Founders of the DuchessBrazil network, they encouraged technical women to participate in the Java community.

Patrycja Wegrzynowicz and **Gail Anderson** are both cofounders of technology companies. Wegrzynowicz specializes in automated software engineering and Java technologies and is a cofounder and CTO of Yonita. She presented on security vulnerabilities in open source Java libraries. Anderson has been designing technical courses and authoring textbooks for more than 20 years and is a cofounder of the Anderson Software Group, a leading provider of software development training. She presented the session “Make Your Clients Richer.”

“We need to stand up for ourselves in our lives and careers,” Senger said. She recommends that women get involved in the Java community by joining user groups and submitting proposals for talks at Java conferences.



Régina ten Bruggencate



Trisha Gee



Yara Senger

New JavaFX Community Site on Java.net

Community activity surrounding JavaFX has been steadily growing, with tweets, blog posts, and projects increasing in number. Now, there is a [JavaFX community site](#) on Java.net. The main purpose of this site is to provide a focal point for the JavaFX community, where relevant tweets, blog posts, and other resources can be easily found. [Gerrit Grunwald](#) and [Jim Weaver](#), the community leaders for this site, welcome your feedback.

GREENVILLE, SOUTH CAROLINA JUG



Java Magazine caught up with [Greenville Java Users Group](#) founder **John Yeary** and five-year Java user group (JUG) member **Glen Peterson** at JavaOne 2012. Greenville, South Carolina, is a small city surrounded by rural communities. Peterson noted that Greenville JUG is “the most interesting thing going on between Greenville and Charlotte” (the closest big city). So, does Greenville JUG have a tailor-made, captive audience? By no means.

Java evangelist [Yeary](#) founded the group in 2002. The first few meetings consisted of up to 3 attendees. Today the group typically draws crowds in the 25-person range.

Growing Greenville JUG was an ongoing labor for Yeary and his fellow members. The fact that the JUG was centered in a primarily rural area created difficulties in contacting people who might want to participate. Yeary experimented with various methods of outreach. His successes included a mailing list for technology managers. These people told their developers about Greenville JUG, and many developers who had been initially encouraged by their managers to attend ultimately became regular attendees at the JUG’s monthly meetings.

A recent turning point in Greenville JUG’s growth was when a [JFrog](#) representative stopped by to participate in a Greenville JUG meeting while on his way to OSCON. The cost of making a stop at Greenville was small compared with the overall cost of the OSCON trip, so Yeary was able to negotiate to get the presentation at the meeting.

Yeary has this advice for JUG leaders who are seeking to find presenters for their JUG meetings: “Be yourself, and introduce the person to your region of the world and to your local developer community—make them feel at home in your locale.” If you do this, word will likely spread among potential speakers that it’s well worth their time to present at your JUG sessions.

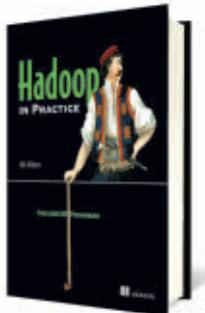
PHOTOGRAPH BY DAVE MCCLINTOCK, SNAPSHOTSC.COM

JAVA BOOKS



[ORACLE CERTIFIED ASSOCIATE, JAVA SE 7 PROGRAMMER STUDY GUIDE](#)

By Richard M. Reese
Packt Publishing (August 2012)
Oracle Certified Associate, Java SE 7 Programmer Study Guide addresses certification exam objectives and provides discussion and examples to show the best ways of applying Java language features in real-world programming. Java SE 7 Associate Programmer certification adds to your qualification as a Java developer. Knowledge of Java is important, but knowing how to write efficient and productive code adds to your skills and gives you an edge. Coverage of the certification objectives goes beyond a simple review, and chapters include sample exam questions.



[HADOOP IN PRACTICE](#)

By Alex Holmes
Manning Publications (October 2012)
Hadoop in Practice collects nearly 100 Hadoop examples and presents them in a problem/solution format. Each technique addresses a specific task you’ll face, such as querying big data using Pig or writing a log file loader. You’ll explore each problem step by step, learning both how to build and deploy that specific solution along with gaining an understanding of the thinking that went into its design. As you work through the tasks, you’ll find yourself growing more comfortable with Hadoop and at home in the world of big data.

2012 JCP AWARD WINNERS



The 10th annual Java Community Process (JCP) Award winners were [announced](#) at the JCP party at JavaOne 2012. The winners were selected from an impressive list of [nominees](#). This year's JCP Member/Participant of the Year Award went to two Java user groups, the [London Java Community](#) and [SouJava](#) (shown above), for their historic contribution to the Adopt-a-JSR program and for supporting Java developers through the JCP. The Outstanding Spec Lead Award went to **Victor Grazi**, for his work on [JSR 354](#), the Money and Currency API. The Most Significant JSR Award went to the [JCP.Next effort](#), led by JCP Chair **Patrick Curran**.



JAVA CHAMPION PROFILE JONAS BONÉR

Jonas Bonér is a Swedish entrepreneur, programmer, speaker, and writer. He became a Java Champion in February 2011.

Java Magazine: Where did you grow up?

Bonér: I grew up in Uppsala, Sweden. Later, I lived in Stockholm, and then in Östersund. Since my family loves skiing, we've also lived in Åre, Sweden; Serre Chevalier, France; Alagna, Italy; and Bad Gastein/Salzburg, Austria, for periods of time—all great ski resorts.

Java Magazine: When and how did you first become interested in computers and programming?

Bonér: I was studying math at the university, but I didn't know what to do with it. First I was thinking of becoming a math teacher. Then I took a CS course just

for fun (Programming in C++), and I realized that that was what I wanted to do. It was applied mathematics, and suited me very well.

Java Magazine: What was your first computer and programming language?

Bonér: I did my very first programming in Pascal on a Solaris terminal at the university; my first computer ran Slackware Linux.

Java Magazine: What was your first professional programming job?

Bonér: My first programming job was as an IT consultant in Uppsala, doing CORBA [Common Object Request Broker Architecture] and EJB [Enterprise JavaBeans] 1.0—what a joy!

Java Magazine: What do you enjoy for fun and relaxation?

Bonér: I enjoy spending



Java Magazine: What are you looking forward to in the coming years?

Bonér: Family-wise, I'm looking forward to seeing the kids grow up; seeing what kind of people they will turn into; and being there for them, listening, and coaching them to materialize their dreams. Business-wise, I'm looking forward to building a successful business around the open source stack we are working on and are passionate about. Personally, I hope to become a bit wiser and better at understanding, selecting, and focusing on the important things in life, the tiny things that make all the difference.

Visit Bonér's company site, [Typesafe](#), and his latest open source project, [Akka](#).

//java nation /



EVENTS

JavaOne Latin America DECEMBER 4–6, SÃO PAULO, BRAZIL

Join Java developers and technologists for Java-focused content, training, and networking. Experts from the worldwide Java community share unique and leading-edge content with attendees. As always, there are keynotes, technical sessions, hands-on labs, demos, exhibitors, and more. Featured content tracks include Core Java Platform; Development Tools and Techniques; Emerging Languages on the JVM; Enterprise Service Architectures and the Cloud; Java EE Web Profile and Platform Technologies; Java ME, Java Card, Embedded, and Devices; and JavaFX and Rich User Experiences. And of course, there are opportunities to network so that you can build your community, share your expertise, and learn best practices.

PHOTOGRAPH BY JEREMY WOODHOUSE/GETTY IMAGES

DECEMBER

JDays 2012

DECEMBER 3–5

GOTHENBURG, SWEDEN

JDays 2012 is a conference about Java, open source, and related technologies. The first two days of the conference feature a lineup of sessions that were voted on by the community. Day 3 of the conference offers a full day of free courses and hands-on training.

Java Conference

DECEMBER 8

BANGALORE, INDIA

The Java Conference is an ideal place to obtain critical skills to help you build, run, and manage tomorrow's software solutions. The speakers are industry practitioners, selected for their knowledge and their ability to

convey that knowledge to Java developers at large. Spend time with hundreds of other Java professionals, share best practices, and take home practical advice that will make an immediate and measurable difference to your projects.

Groovy and Grails eXchange 2012

DECEMBER 13–14

LONDON, ENGLAND

This two-day conference features two dozen expert-led talks, along with discussion and brainstorming sessions, all focused on learning and sharing ideas, tools, and best practices for enterprise Web development with Groovy and Grails.

7th Annual IndicThreads Pune Conference

DECEMBER 13–15

PUNE, INDIA

This conference explores a wide array of software development tools and technologies in Java; cloud computing; mobile application development; and emerging technologies such as big data, gamification, HTML5, and more.

JANUARY

Take Off

JANUARY 17–18

LILLE, FRANCE

This English-language conference for Web developers and designers focuses on trending topics. Session content ranges from server side to pure front end, from design and philosophy to new languages, and from frameworks to development techniques.

FEBRUARY

Jfokus 2013

FEBRUARY 4–6

STOCKHOLM, SWEDEN

Jfokus is the largest annual conference for everyone who works with Java in Sweden. It is arranged in collaboration with Javaforum Stockholm, a Swedish developer community and a Java user group. Over three days, get up to date on the latest developments in the Java platform. The agenda includes rock-star speakers, both from Sweden and around the world, with a focus on systems development with Java and surrounding techniques such as dynamic languages and agile methodologies. Get the latest trends and buzz about Java from people who live and breathe technology.

(open source)



Continuing our series of interviews with distinguished members of the Executive Committee of the Java Community Process (JCP), we turn to *Mike Milinkovich, executive director of the Eclipse Foundation, which was created in January 2004 as an independent not-for-profit corporation to foster a vendor-neutral, open, and transparent Eclipse community. Historically, Eclipse became famous as a Java IDE and a plug-in-based platform for building software development tools.*

JCP Executive Series

A CONVERSATION WITH MIKE MILINKOVICH

The Eclipse Foundation's **Mike Milinkovich** assesses the state of Java and the JCP. **BY JANICE J. HEISS**

ART BY NICHOLAS PAVKOVIC, PHOTOGRAPHY BY BLAIR GABLE/GETTY IMAGES

Eclipse Foundation Executive Director Mike Milinkovich meets with a colleague in his Ottawa, Ontario, Canada, office.



The Eclipse community now has more than 200 projects, 190 organizational members, and a wealth of interesting technologies that go far beyond its beginnings as a Java IDE. The Eclipse Foundation's mandate is to ensure that its projects are ready for commercial adoption. As such, its governance model, development process, and IP management are tailored to make sure that consumers and adopters of Eclipse

technologies have a say in the Eclipse Foundation.

Milinkovich has a long and diverse history with Java that enables him to grasp a variety of perspectives. He first worked with Java as IBM's VisualAge for Java strategy manager in 1997. In 1999, he left IBM to work at Object People. In addition to its strong involvement in Java training and consulting, Object People had a persistence product,

TopLink, which was acquired by a tools company named WebGain in 2000. Then in 2002, TopLink was acquired by Oracle, where Milinkovich served as vice president of Oracle application server technical services. In 2004, he assumed his present position where, among his many duties, he represents the Eclipse Foundation on the Executive Committee of the JCP.

Java Magazine: Tell us about your day job.

Milinkovich: I've had jobs where I was a pure developer, and I've had jobs where I was a pure business person. And I've always been happiest when I'm oscillating back and forth between the two—and this job has this in spades.

So basically I spend a lot of time keeping a pulse on what's going on in the Eclipse projects and making sure the projects are well supported. We may help project leaders by introducing them to others in the community or making sure that their code repositories and bug trackers are up and running. On the business side, Eclipse is also set up as a consortium. So we have a lot of member companies that are trying to build a business or products on top of Eclipse. We help them move their business forward. Then there's always both member and project recruitment.

So I spend a lot of time talking to people in lots of different technology



Milinkovich is happiest mixing development with business, and his Eclipse Foundation position offers just that mix.

domains about why they would want to do open source at Eclipse. We talk about how they can use Eclipse to leverage their business models and move forward. I have been in this job now for more than eight years, and I still really enjoy it.

Java Magazine: In January 2007, Eclipse joined the JCP. At that time, you said, "Any changes we would be proposing to the JCP would center around making the process more open and transparent." How have things gone in the subsequent five-and-a-half years?

Milinkovich: I think it's gone fairly well. The new JCP 2.8 process document requires all JCP expert groups to op-

erate in a transparent manner. That's a big step forward compared to six years ago. And looking at Java as a whole, the Java platform is becoming more open, with the basic platform work happening at OpenJDK and with the participation of additional companies.

Java Magazine: Do you have any comments about the new presence of Java user groups on both the Executive Committee and the JCP in general?

Milinkovich: I think it's great. Specifically, the presence of Ben Evans and Martijn Verburg from the London Java Community and Bruno Souza from Brazil's SouJava has been a very positive influence. They bring the perspective of Java developers to the table, which I think has been somewhat lacking in the past. Their voices are very important. In addition, in the last couple of years, what I call "Java user companies"—I'm thinking of Credit Suisse and Goldman Sachs—have also joined the Executive Committee, bringing a very helpful perspective to the conversations as consumers. They're in

the position of trying to use this technology for their customers' needs and their business needs. That's an important viewpoint.

Generally, broadening the perspectives that you have in your governance model helps any community. We see that at Eclipse."

nity. We see that at Eclipse, and I think it's equally true at the JCP.

Java Magazine: What is the best thing that has happened at the JCP in the last few years?

Milinkovich: It's really simple—we got moving again. During the last couple of years with Sun at the helm, it was obvious that there wasn't a lot of investment going on in Java. There was also the stalemate regarding moving forward with Java 7 at the JCP Executive Committee level. And since Oracle has taken the helm, we got going again.

Java 7 was shipped, and Java 8 and Java 9 are underway. And within the JCP itself, there have now been two revisions of the process documents, bringing the JCP 2.8 process forward, and then we have a second JSR to merge the two executive committees into one [JSR 355]. We are also starting another JSR to rewrite the JSPA [Java Specification Participation Agreement] itself. So there's been lots of val-

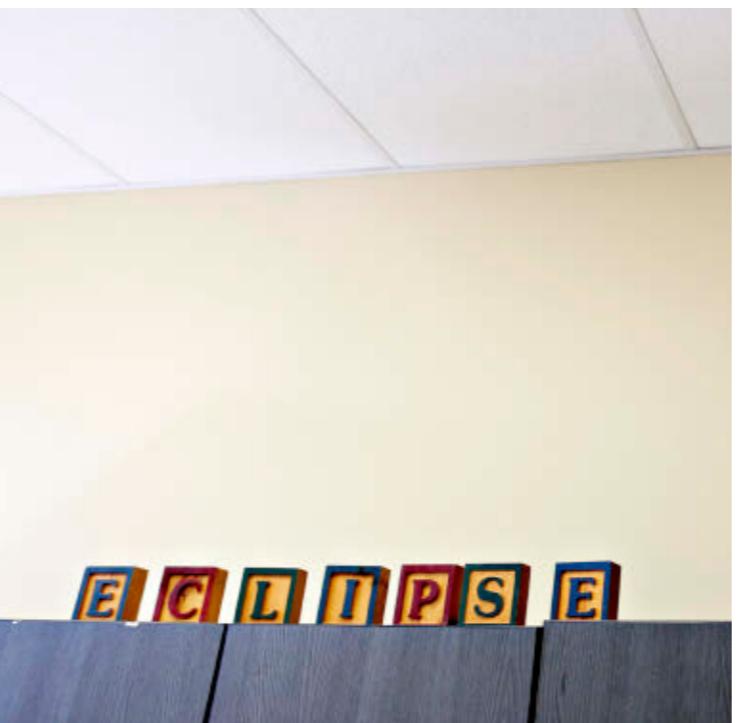
uable work in bringing Java forward.

Java Magazine: In walking the fine line between respecting standards and encouraging innovation, does the JCP err too far in one direction or the other?

Milinkovich: Yes and no. The JCP has a lot of different JSRs underway, and there are a lot of different

OPEN VIEWS

"Broadening the perspectives [of] your governance model helps any community. We see that at Eclipse."



Left: Milinkovich analyzes some development details with Wayne Beaton, director of open source projects at Eclipse. **Right:** Spelling things out.

things going on. So there are areas where I think they get it right and some where it's not so good.

First, innovating via standards is an inherently bad idea. Successful innovation is inherently about seeing which ideas are truly useful. Standardizing technologies that haven't already proven themselves in the marketplace means that bad ideas inevitably leak in. To cite my favorite example, compare the adoption and success of Spring versus CMP [container-managed persistence]. So I think the JCP needs to focus on standardizing innovations that have already happened as opposed to trying to inspire

innovation in the standards process. That's particularly true of bringing in very new technologies and adding them to the Java platform.

Now, on the flip side, the JCP process is also the mechanism by which the Java platform has evolved. And I think the more open and community-based that we can make the specifications and the implementations, the better the platform is going to be.

So in this context, the recent direction of closely tying the work in the JCP with what's going on in the OpenJDK implementation is very positive. There's a process right now of getting some of the spec work for the Java 8 JSR 337 done

IDEAS FIRST
"Innovating via standards is an inherently bad idea. Successful innovation is about seeing which ideas are truly useful."

on an OpenJDK mailing list—that's a really good thing.

Java Magazine: Do you have any ideas about how Oracle should best negotiate the inevitable tension between running a business and leading an open source Java community?

Milinkovich: Let me offer a lesson from the Eclipse experience. Historically, Eclipse started as an IBM project and initially, there was lots of tension between IBM and Sun Microsystems about the future of Java and how Java should be governed. And what IBM tried in part to do when they set up the Eclipse Foundation was demonstrate how they thought Java should be governed. The lesson learned was that if you truly set it free at arm's length, good things can happen. The Eclipse Foundation today has nearly 190 organizational members. There are literally thousands of products that are built on top of Eclipse. And to a considerable degree, that happened after IBM let Eclipse go and set up the Eclipse Foundation as an independent entity.

I'm not sure if Oracle could ever do that, in part because of the business legacy of Java and the way it was initially established. But the more it can make the technology open to all comers and operate through a transparent and meritocratic community-based approach, the more successful it will be.

One area of tension between the success of Java and Oracle's business



Magazine covers that have featured Eclipse line the walls in a hallway at the Eclipse Foundation.

needs is in the field-of-use restrictions that are holding Java back from success in the embedded and mobile spaces. At some point—I don't know when or how—Oracle is going to have to get rid of them because in today's world, if you don't have both a technology and business adoption strategy in mobile and embedded, you're dead. I don't know the solution, but Oracle will definitely need to change this in order to make Java competitive with Android, to give but one example.

Java Magazine: Finally, what could

OpenJDK learn from the Eclipse community?

Milinkovich: OpenJDK recently announced that they were going to adopt Eclipse's approach of shipping regular, predictable release trains. I think that Eclipse has shown that predictable release schedules are very helpful to the ecosystem. So that's one lesson that's been shared already.

But in comparing OpenJDK to Eclipse we need to understand that Eclipse is truly an independent organization, set up as a separate corporation with

a board of directors that oversees the entire organization, sets the budget, sets the strategy, and so forth.

OpenJDK is, like the JCP, effectively an arm of Oracle, which puts certain constraints on the way things get done. All the intellectual property flows to Oracle and, so far, the infrastructure for OpenJDK has been constrained by the ability of Oracle to put resources into improving it. So, for example, it's taken quite some time—and it's still not done—to get an open, non-Oracle bug database available for the whole OpenJDK community.

What can OpenJDK learn from Eclipse? I think that, to the degree possible, being truly vendor neutral in a level playing field across all of the companies and individuals involved is absolutely key. The degree to which OpenJDK can bring in other companies, participants, and contributors will be the best measure of success. OpenJDK has certainly done well in the last year and a half, bringing in Red Hat, IBM, SAP, and a few others as well. That's a big step forward for the OpenJDK community. </article>

Janice J. Heiss is the Java acquisitions editor at Oracle and a technology editor at *Java Magazine*.

LEARN MORE

- [Mike Milinkovich's blog](#)
- [Follow Milinkovich on Twitter](#)

Data Quality Tools for Java



Now, finding the right data verification tools doesn't have to be so puzzling. Melissa Data offers customizable APIs, Web services and enterprise applications to match your budget and business needs. For solutions to cleanse, validate and standardize your contact data, we're ready to help you find the perfect fit.

Multiplatform

Request free trials at
MelissaData.com/myjava or call 1-800-MELISSA

- Global address verification for 240 countries
- Clean and validate data at point-of-entry or in batch
- Correct misspellings, missing directionals, and confirm deliverability
- Enhance addresses with County, Census, FIPS, etc.
- Append rooftop lat/long coordinates to street addresses
- Update records with USPS and Canadian change of address info

MELISSA DATA®
Your Partner in Data Quality



(open source)

GROWING ON OPEN

AgroSense provides an all-Java open source platform for sustainable farming and precision agriculture.

BY PHILIP J. GILL

Sun, soil, seeds, water, fertilizer—for centuries these have been the essential ingredients for successful farming. But with the world's population at 7 billion and growing; arable, fertile land at a premium, and environmental degradation a genuine concern, farmers around the globe are looking to increase crop yield and quality while conserving resources—all in an environmentally friendly and sustainable way.



ART BY NICHOLAS PAVKOVIC, PHOTOGRAPHY BY TON HENDRIKS

**Timon Veenstra,
Project Leader of
the AgroSense
Project**

SNAPSHOT

THE AGROSENSE PROJECT

agrosense.org

Industry: Agriculture

Java version used:



Veenstra and members of the AgroSense Project team in the Ordina offices in Nieuwegein, the Netherlands

To do this, farmers—from small family enterprises to giant, multinational agribusinesses—are looking to add a new ingredient to the mix: information.

"Farming today is a very information-dense industry," explains Timon Veenstra, project leader of the AgroSense Project. "There's informa-

ion about the soil, the soil type, the complexity of the soil, its ingredients, the nutrients present. There's information from satellites about weather and weather predictions. And then there's information that farm machinery collects for every GPS coordinate in the field—what direction was the

tractor going, what was the height of the mower, what was my plow depth, things like that."

For too many farmers, however, all that information exists as raw data with little meaning. "Farmers have lots and lots of data, but they have little information," says Veenstra. "So much



so that a lot of that data gets thrown away because they lack a platform to process that data into information."

Providing a suitable platform for just that task is the goal of the AgroSense Project, an open source farm management system being built on Java and the NetBeans Platform. Veenstra launched AgroSense—originally named “Cloudfarming”—in June 2011 with initial support and funding from [Ordina](#), an IT consulting and outsourcing firm based in Nieuwegein, the Netherlands, where he works as a software architect and developer. Although Veenstra is focused on agriculture in the Netherlands right now, he says he sees no reason why the capabilities he envisions for the AgroSense platform would not be applicable to farmers around the world.

“The goal is to make everything a farmer does available on the AgroSense platform,” says Veenstra.

BIG BUSINESS
Farms cover more than 56 percent of the arable land in the Netherlands and employ 1 in 10 workers.

The central component of the AgroSense platform is an onscreen map that displays not just the topography of the land, but also all the critical information that farmers need to know.

AGRICULTURAL POWERHOUSE

The Netherlands may be a small country, but it's one of Europe's and the world's largest economies and most prosperous countries. Besides being home to large industrial organizations such as Philips Electronics NV and Unilever and oil-and-gas giant Royal Dutch Shell, the Netherlands is the third-largest exporter of agricultural products in the world, behind the United States and France. More than 56 percent of the country's arable land is under cultivation, and agriculture provides 1 in 10 jobs and generates 10 percent of the gross national product. Its leading crops are tomatoes, cucumbers, chilies, and fresh-cut plants and flowers—the country has of course been famous for its tulips for centuries.

“Because the country is so densely populated, agricultural land is very scarce and very expensive,” says Veenstra. “Farmers are sitting on millions of euros of land value, so they are eager to get the most value in crops from their land.”

This has also made Dutch farmers open to a rela-

Precision Agriculture

Precision agriculture, also called *precision farming*, is a relatively new farm management concept that many see as the future of sustainable farming.

Precision agriculture differs from traditional farming practices in that it enables farmers to manage their fields at an intrafield level. In traditional agriculture, a farmer would just take a basic value of fertilizer or pesticide, for instance, and apply it to all of the land. “This approach will always be too much, especially for pesticides,” explains Timon Veenstra, project leader of the AgroSense Project, an open source software platform for precision agriculture. “While there is disease on the land, it's usually just on one part of the land, but farmers would apply the pesticides to their entire land.”

That's not just a waste of expensive resources and pesticides; it's also potentially damaging to the land and the environment, notes Veenstra.

And too much fertilizer can result in poor-quality crops. “If you apply too much fertilizer, the crops will grow too fast and be too weak, so when the winter winds come they'll fall down,” he says.

But with precision agriculture, Veenstra continues, “farmers use information from sensor values and soil samples taken from each part of the field and only need to spray pesticides on the affected areas and to apply fertilizer only to the part of the land that needs it.”

Although precision agriculture originated in the United States in the 1980s, interest has spread worldwide, including to the Netherlands, which is the world's third-largest exporter of agricultural products, behind the United States and France.

What's Driving Open Source?



 Simon Phipps, president of the Open Source Initiative, talks with *Java Magazine*'s Tori Wieldt about what's driving open source today.

tively new concept in farm management called *precision agriculture*. Also known as *precision farming*, it combines crop science, responsible environmental stewardship, and efficient farm practices so that farmers can grow more crops of higher quality while consuming fewer resources and minimizing environmental impact and footprint (see sidebar, "Precision Agriculture").

"With precision agriculture, farmers can manage the land down to the square-meter level, rather than field by field," explains Veenstra. "Precision agriculture makes farmers more productive, enabling them to produce higher quality and a higher quantity of crops."

AgroSense—a [2012 Duke's Choice Award winner](#)—is a modular farm management system for precision agriculture. The central component of the AgroSense platform is an onscreen map that displays not just the topography of the land, but also all the critical information that farmers need to know, such as soil ingredients, crop types, fertilizers and pesticides used, and much more. AgroSense uses satellite images, JPEGs, and open source and proprietary mapping services to create maps, and even allows farmers to draw their own.

The information is displayed in the context of whichever module or application is being used, explains Veenstra. "If you're operating in the fertilizer module, for example, when farmers

click on the field they will see all the information about fertilizers being used on that field, how much is being used, even if the fertilizer is in stock," he says. "And if it's not in stock and they need more, the platform will connect farmers to an online catalog so they can order it."

"When the farmer changes the context to do something else—say, plan a new crop rotation—then the entire context will switch to that new module," he adds, "and it will display the information and the appropriate tasks."

In its first release, slated for late 2012, the AgroSense platform will focus on crop planning. "With the tools

in the first release, farmers will be able to plan their crops for a year," says Veenstra. "They will be able to import a shapefile with the digital representation of their fields, or they can draw new fields on their map. Then they can select the crop they want to grow and which periods they plan to grow it in."

This first release will be geared toward the farm management home office, which usually has good internet connections, says Veenstra. In later releases, Veenstra envisions that farm workers on tractors will have a variety of mobile devices, such as cell phones and tablets, either mounted on tractors or carried in their pockets, that will



Veenstra and AgroSense Project team members catch up over coffee.

collect data in the field and transmit it to the main office.

OPEN PLATFORM

Veenstra chose to go the open source route with AgroSense for several reasons. First, such a comprehensive project is beyond the ability of a single programmer or even a single company. Open source makes it accessible to as many programmers and developers as possible, says Veenstra, who plans to open an online app store similar to Apple's to help round out AgroSense's capabilities. "We are looking to partner with various manufacturers and others to provide links to their resources," says Veenstra. "We also expect that different manufacturers and software developers will build their own applications that they can plug into the AgroSense platform."

Another reason was his choice of software. AgroSense is a rich-client platform built in Java on the NetBeans Platform. Although alternatives were considered, says Veenstra, "NetBeans is highly modular and really solid." Building Java software on top of the NetBeans Platform can save years of development time—the platform is a generic framework for JavaFX and Swing applications, and it provides the "plumbing" developers would otherwise have to write them-

selves, such as saving state; connecting actions to menu items, toolbar items, and keyboard shortcuts; and window management. The NetBeans Platform consists of a reliable and flexible application architecture and provides a time-tested architecture for free—and it's also an architecture that encourages sustainable development practices. Because the NetBeans Platform architecture is modular, it's easy to create applications that are robust and extensible.

Veenstra also uses the NetBeans IDE and [Apache Maven](#), an open source software project management tool. The combination of the modular NetBeans Platform and Maven is very powerful, he says, noting that the Maven build structure allows for easy reuse of modules in other applications. "There are already NetBeans Platform applications using AgroSense modules, such as the map viewer, and easy reusable components is one of the key factors of a successful develop-

ment ecosystem," Veenstra adds. "Every new reusable module means a bigger head start in development of new applications. The NetBeans IDE supports Maven very well, and it is extended with every new NetBeans release. The great toolset that the NetBeans IDE provides allows for very high development productivity."

GOING OPEN

Open source makes AgroSense accessible to as many programmers as possible.

For Veenstra, another aspect of NetBeans' appeal is the fact that it's more than a technology platform; it's a vibrant, vital community with active users worldwide who are always willing to pitch in and answer questions to help out fellow users. The NetBeans IDE has more than 1 million active users worldwide, and there are at least hundreds of applications created on top of its framework.

But Veenstra notes that the main reason to go with open source for AgroSense is that open source projects take on a life of their own—far beyond the funding of one company or the efforts of one programmer. And the goals of the AgroSense Project, he says, are too important for it to wither.

Although Ordina sponsored and funded AgroSense's initial development, the company is looking to partner with another firm to continue development. "As an outsourcing firm, product development is not in line with our corporate strategy," says Veenstra.

"AgroSense is and will continue to be an open source project," he continues. "Although AgroSense's development speed will be influenced by the involvement of sponsoring companies, its continuation won't be." </article>

Philip J. Gill is a San Diego, California-based freelance writer and editor who has been following Java for 20 years.

Hadoop creator
Doug Cutting (left)
with members of
the Cloudera team



ART BY NICHOLAS PAVKOVIC, PHOTOGRAPHY BY BOB ADLER

(open source)

AN ENGINE FOR BIG DATA

Hadoop uses Java for large-scale analytics.

BY DAVID BAUM



Whether it's searching for subatomic particles from an atom smasher or discerning purchasing patterns in billions of Web pages, big data presents a vast array of complex computational problems, stemming from the constant influx of digital information. The world created more than



SNAPSHOT

CLOUDERA

cloudera.com

Headquarters:

Palo Alto, California

Industry:

Enterprise software

Employees:

300

Java technology used:

Java SE

1.8 zettabytes (1.8 trillion gigabytes) of information last year, according to International Data Corporation's *2011 Digital Universe Study*, fueled by billions of mobile phones, tens of billions of social media posts, and an ever-expanding array of networked sensors from cars, utility meters, shipping containers, shop-floor equipment, point-of-sale terminals, and many other sources. Thanks to Apache Hadoop, organizations now have a cost-effective way to analyze all this data.

Hadoop, a *2012 Duke's Choice Award winner*, is the brainchild of Doug Cutting, chief architect at Cloudera

and chairman of the [Apache Software Foundation](#). Named after a toy elephant that Cutting's children used to play with, this open source software platform, written in Java, enables businesses to unlock potential value from their data using inexpensive commodity servers. Sometimes described as a distributed operating system, Hadoop makes thousands of computers appear as one unified system for storing data and running computations. It's an ideal way not only to analyze structured data from enterprise applications, but also to work with unstructured information from Web pages, social media sites,

Members of the Cloudera team brief Cutting on their current projects.

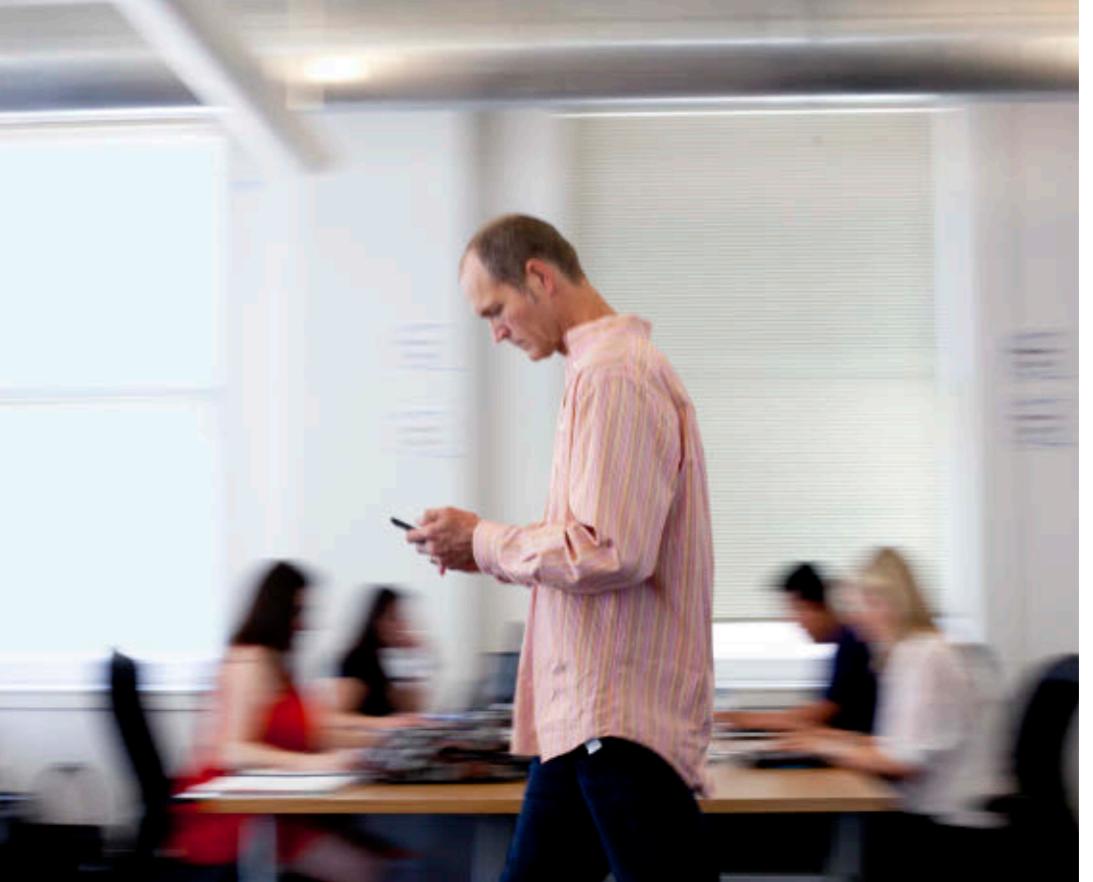
e-mail exchanges, search indexes, clickstreams, equipment sensors, video cameras, and more. Hadoop can easily scale to accommodate more capacity simply by adding more servers and storage equipment. Its distributed nature and parallel processing architecture ensure a high-availability infrastructure.

"Hadoop is based on using commodity computers to construct a distributed system that is tolerant to failure, so if you lose a node everything will continue operating," Cutting explains. "If you want to crawl 10 billion Web pages each month without having to buy some sort of exotic hardware, then you need a bunch of computers working in parallel. We are talking about much more data than you can store on an array of hard drives, and far more than you can process in a reasonable amount of time on a single server—at least an affordable server."

Achieving this level of performance and reliability at a reasonable price point is appealing to all types of organizations. Credit card companies use Hadoop to help them predict fraud. Energy companies use it to search for oil deposits. Social media sites such as Facebook and LinkedIn use Hadoop to personalize interactions between their members. Online dating companies



Left: Cutting works through lunch as he prepares for an afternoon meeting. **Right:** Cutting buzzes through the Cloudera offices.



use it to make the perfect match based on a dizzying array of preferences from millions of singles. And many commercial organizations use it to gauge customer sentiment.

As different as these applications may seem, they all have one thing in common: the need to find patterns in immense volumes of information. Hadoop takes computational tasks that were formerly relegated to expensive supercomputers and spreads them out over thousands of commodity servers and storage devices.

FROM RESEARCH TO PRODUCTION
Well-known in the open source community as the founder of the [Lucene](#), [Nutch](#), and [Avro](#) projects, Cutting is an expert in information retrieval and full-text search. After earning a degree in linguistics from Stanford University in

1988, he went to work at the Xerox Palo Alto Research Center (PARC), where he developed a high-performance retrieval engine, several innovative search paradigms, advanced linguistic analysis methods, and high-quality text summarization algorithms. His work at PARC resulted in six patents in the fields of text searching, retrieval, browsing, and indexing.

After leaving PARC in 1993, Cutting held positions at Apple, Excite, and Yahoo! before venturing into open source in 2000 with the Lucene project, an efficient, full-featured text search engine written in Java. His work in the open source community has proceeded in tandem with his commercial ventures. At Yahoo! he was a

key member of the team that built and deployed a production Hadoop storage and analysis cluster for mission-critical business analytics. He joined Cloudera in 2009 to help release [CDH](#), a 100 percent open source Hadoop distribution, built specifically to meet enterprise demands.

Thanks to the efforts of Cutting and hundreds of other developers, Hadoop is now broadly accepted throughout the technology world as a good choice for crunching lots of data in large production settings. Facebook claims to have the largest Hadoop cluster, with 30 PB of data. Yahoo! runs its Search Webmap as a Hadoop application on more than 10,000 core Linux clusters. Google,

GROUP EFFORT
Hadoop is not controlled by any one institution, but by different institutions working in collaboration.

Joost, Microsoft, Netflix, Twitter, and the *New York Times* also depend on Hadoop for large-scale computing tasks.

OPEN COLLABORATION

Derived originally from Google's MapReduce and Google File System papers, Hadoop has a colorful history that illustrates the interplay between today's commercial and open source interests.

"I was working with a small team on a project called Nutch to create a Web crawler and indexer—similar to Google and Bing, but as an open source project," Cutting recalls. "When Google published a paper about its distributed file system and the MapReduce computing layer that goes on top of it, we could see that this was what we needed: a systemic way to spread storage and computing across a set of machines that wouldn't be terribly difficult to program and operate."

Cutting decided to rewrite those methods in Java as open source and include the methods in the Nutch project. When Yahoo! expressed interest in the distributed parts of the file system, Cutting split those methods out of Nutch, renamed them Hadoop, and began developing Hadoop as a separate venture.

Today, many organizations contribute to the Hadoop ecosystem. This diverse worldwide community attracts independent developers, contractors, commercial ventures, and government

organizations. Yahoo! has been the largest contributor to the project and continues to use Hadoop extensively across its businesses.

"Hadoop is not controlled by any one institution, but rather by all these different institutions working in collaboration," Cutting explains. "Decisions about the software are made by consensus. You can't just do things unilaterally."

In order to explain the Hadoop ecosystem, Cutting often compares it to Linux. "Linux is more than just the kernel," he says. "Just as there is a whole style of programming and interaction that is built on top of Linux, developers in the Hadoop ecosystem build applications that share essential coordination features, such as security."

ADVANTAGE: JAVA

According to Cutting, developing a distributed software platform is an order of magnitude trickier than developing software that only runs on one machine. Fortunately, Java simplifies the process on many levels.

"At each point that you interact with something remotely, you're not sure what can happen," he explains. "There can be delays, you can have lots of strange timing issues, and there could be failures, which means you can have errors popping up in places that you didn't expect. The more distributed the system, the more these effects are compounded."

Because debugging a distributed system is much more difficult than developing programs on a single computer, it makes sense to establish a common OS layer at the foundation. Java pro-

Industry Applications for Hadoop

E-Tailing

- Recommendation engines for cross-selling
- Ad-targeting, analysis, forecasting, and optimization
- Event analytics (determining what series of steps led to a desired outcome)

Financial Services

- Compliance and regulatory reporting
- Risk analysis and management
- Fraud detection and security analytics

Healthcare and Life Sciences

- Patient care quality and program analysis
- Supply chain management
- Drug discovery and development analysis

Retail/Consumer Packaged Goods

- Merchandising and market basket analysis
- Campaign management and customer loyalty programs
- Supply chain management and analytics

Telecommunications

- Customer churn prevention
- Call detail record (CDR) analysis
- Network performance and optimization

Government

- Cybersecurity
- Compliance and regulatory analysis
- Energy consumption and carbon footprint management

vides the right combination of easy debugging and good performance.

"Type safety and garbage collection make it a lot easier to develop new systems with Java," Cutting notes. "You don't have to worry about memory leaks. And Java programs tend to crash less catastrophically. You can check a lot of things at compile time. If a program does fail at runtime, it is easier to debug in Java than it would be in C or C++."

Cutting values the rich ecosystem of Java libraries, utilities, and tools, which give developers choices for how they wish to develop, share, and maintain their Java applications. "In the open source community, we have a large

library of tools. The libraries available to Java developers are more extensive than for any language I've seen."

Finally, Java's broad acceptance and inherent portability make it easy to develop standard programs that run interchangeably across different operating systems, such as the many 64-bit and 32-bit versions of Linux and Windows. This flexibility was essential to the creation of [Hadoop Distributed File System](#) (HDFS), the primary storage system used by Hadoop applications, which was written in Java for the Hadoop framework.

"With Java, you can basically write once and run anywhere," Cutting emphasizes. "It lets us deliver results not just reliably but quickly. On top of that, Java is a relatively simple language to learn. It's fairly constrained, while still being powerful. It is easier to program with best practices in Java simply because there aren't many things you can do that would make the software very difficult to maintain. That helps when you're developing a large, collaborative project."

INTERACTIVE FUTURE

Hadoop grew up as a platform for batch computing. This was due, in part, to

the nature of the MapReduce computing metaphor on which it is based. Most Hadoop jobs include a large set of input files that generate a large set of output files. Queuing these jobs to run in batch mode allows organizations to process massive amounts of data. In the future, however, Cutting foresees Hadoop evolving to support interactive computations and incremental updates based on a variety of new query engines that have been added to the ecosystem.

"You can now do simple queries and get results while you are waiting," he explains. "I think this platform is going to become more and more general-purpose. It will support new kinds of applications that aren't based on batch computations and aren't just simple key value stores. This means you will be able to submit fairly complex queries over complex tabular data in ways that are much more scalable than traditional solutions."

Whether it's a manufacturing company monitoring data from sensors on the shop floor, an advertiser studying location data to target consumers, or an "e-tailer" examining Web log files to improve customer segmentation, Hadoop has a solution. <[/article](#)>

Cutting takes a break to check e-mail in the lobby of Cloudera in Palo Alto, California.



Based in Santa Barbara, California, **David Baum** writes about innovative businesses, emerging technologies, and compelling lifestyles.

//new to java /



Part 3

Can You Teach Testing to Beginners?

BlueJ melds interactive and regression testing, making testing a reality even for beginners.

MICHAEL KÖLLING



BIC



In the previous issues of Java Magazine we had a look at the BlueJ environment and how to use its unique tools to develop a small application. In this article, we will concentrate on one very specific aspect of BlueJ: its support for testing.

As we said before, BlueJ was designed specifically to support the learning and teaching of programming.

Its tools are targeted at illustrating object-oriented programming concepts, and the emphasis generally is on the development of small-scale projects. There are, however, a few tools in BlueJ that I sometimes wish I had available when working on large, professional projects; I wish the Eclipses and NetBeans of this world would take a slice of this toolset and offer

something similar. BlueJ's interactive testing support is one of these tools.

As we will see, interactive testing with optional test recording is a great way to teach beginners about testing, but it is also occasionally useful for seasoned professionals. But let's start at the beginning.

For this session, we will use a variation of the “people” project that we saw earlier. This version is called people-test. Download and unzip this file, and open it in BlueJ to play along. **Figure 1** shows the main window after you open the project.

TEST ASAP

One of the great features of BlueJ is that you can interactively test any method as soon as it has been completed.

or students—in a “database” (well, a simple collection, really, but that’s OK for now). Its class diagram tells us that there are five classes involved, and that a **Database** holds **Person** objects. It is currently in a state where the implementation has been completed (according to the programmer) and our job is to test whether it works.

Interactive Testing

One of the great features of BlueJ is that you can interactively test any method as soon as it has been completed. Right-click the **Student** class and select the first constructor (see **Figure 2**) to interactively create a **Student** object.

The object will then appear

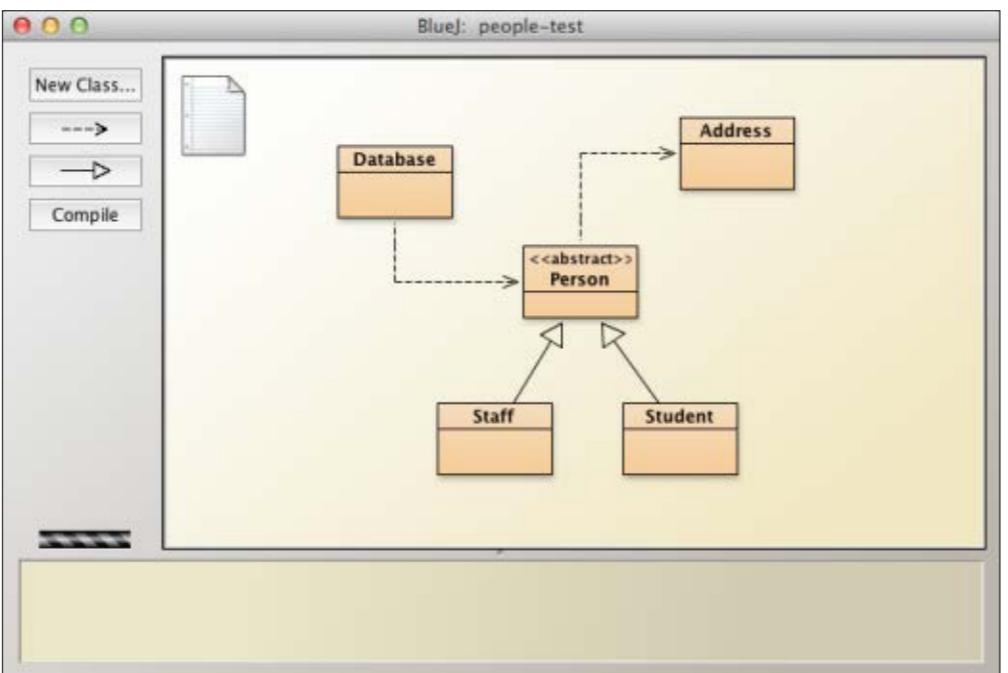


Figure 1

PHOTOGRAPH BY JOHN BLYTHE

//new to java /

on the object bench, where you can right-click it to interactively run any of its public methods (see **Figure 3**).

This technique was described in more detail in the July/August edition of *Java Magazine*. You will see that you can create objects by invoking constructors, easily call any method, pass parameters, and receive return values. You can also compose objects, that is, pass one object as a parameter to another.

Play around with this a bit: Create some **Student** objects with different parameters and try their methods. Create a **Database** object and enter a few **Staff** and **Student** objects to it.

Testing as a Teaching Tool

The possibility to interactively test your code has great benefit as a teaching tool.

Most obviously, you don't need to write any test drivers to test your code. This makes testing much quicker and easier. We always encourage our students to test any method as soon as it has been written.

LEARNING BOOST

The possibility to interactively test your code has great benefit as a teaching tool. It forms a strong mental model of object orientation.

There is no need to complete a whole project; as soon as a single class compiles, we can test any of its methods.

Another aspect of this interactive invocation is even more important: it forms a strong mental model of object orientation. By manually playing through the creation of objects and the invocation of methods, students develop an almost natural view of an application as defined by a set of classes that can be instantiated into objects. The class/object relationship becomes clear, and it quickly seems natural that we communicate with objects by invoking their methods. The concepts of parameters, return values, types, and state can also all be immediately experienced. As a teaching tool, this is one of the most powerful aspects of BlueJ.

The Bug

If you played around with some **Student** objects, and you paid attention to the comments, you may have noticed a bug in our **Student** code. The comment for the **getAccountName** method stated that the account name for a student should consist of a string constructed from the first three letters of the student's name, followed by the first four digits from the student's ID number. Thus, if

we have a student named James Duckling with student ID 123456, the account name should be Jam1234. However, when we call the **getAccountName** method, we can see that the result is incorrect. (This is not the only problem here. We run into even worse trouble if the name or the account name is too short to extract the substring, but I'll leave that as an exercise for you.)

The first lesson here is this: interactive testing, if done regularly, can uncover bugs early. That's good.

There is, however, one disadvantage to interactive testing: to be valid, it has to be repeated after every code change, and the tests accumulate (and, therefore, become more tedious) over time. It's time for a regression testing tool.

Regression Testing

BlueJ supports regression testing via integration of the ever-popular **JUnit** tool. JUnit allows us to write test cases that can be replayed easily, and regression testing suddenly becomes manageable. (In this article, I will not try to explain JUnit or its terminology. I assume you know about JUnit already, or you can easily find out using one of several good tutorials available online.)

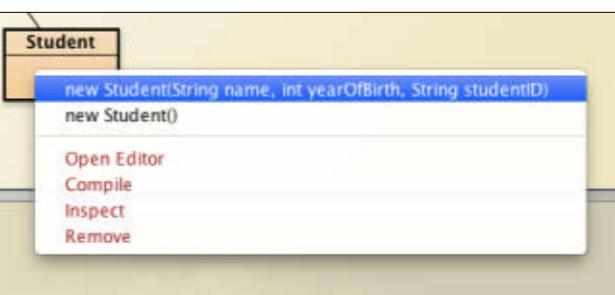


Figure 2

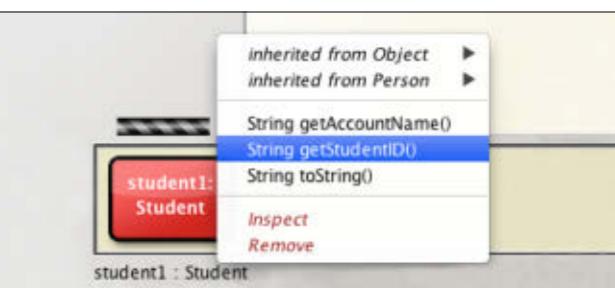


Figure 3

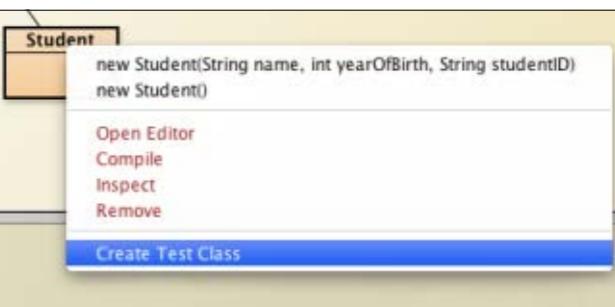


Figure 4

To keep BlueJ's interface simple, the testing tools are initially hidden. They can be switched on in the preferences by going to the **Miscellaneous** tab and selecting the **Show unit testing tools** option. Once testing has been enabled, BlueJ offers some more buttons in the left toolbar and also an additional option in the class's pop-up menu: **Create Test Class** (**Figure 4**).

If we use this function, a test class is created and attached to the

//new to java /

reference class; it is shown in the diagram in green. When we drag our classes in the diagram, the test class will always be kept close to the reference class. **Figure 5** shows our project with two test classes created: one for the **Student** class and one for the **Database** class.

We can now open the editor for the test class, and we see that it contains an automatically generated skeleton for a standard JUnit class. If we are familiar with JUnit, we could now start writing JUnit test cases in the standard JUnit way. This is a big improvement over repeated interactive testing, and it is very useful. However, it gets *really* interesting when we start combining interactive testing with JUnit.

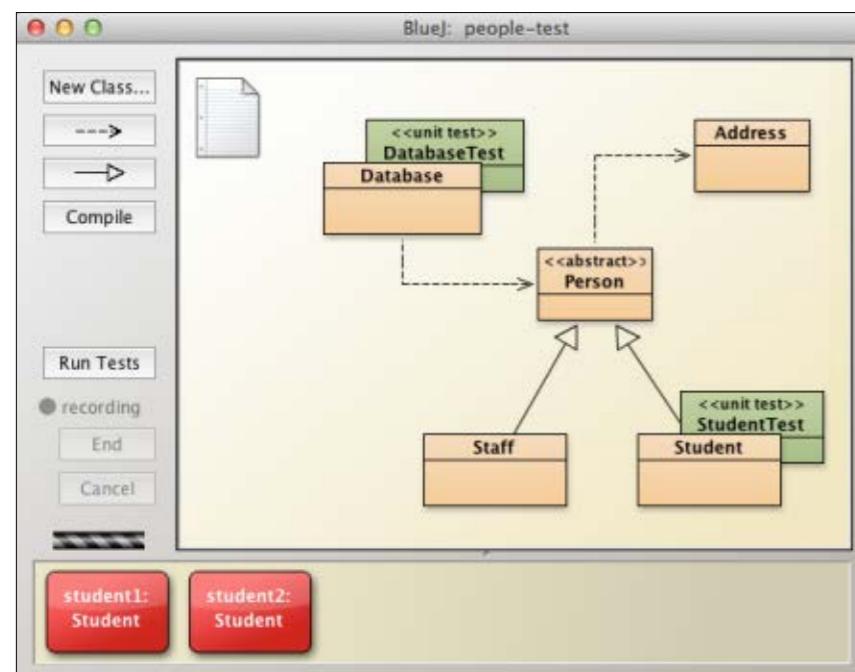


Figure 5

Test Recording

In BlueJ, you not only have a choice between interactive testing and writing JUnit code, but you can combine both. You can record your interactive tests to get them translated automatically into replayable JUnit code. Let's play this through by creating a test case for the bug with the incorrect account name, which we discovered above.

From the test class's pop-up menu, choose **Create Test Method**, as shown in **Figure 6**. BlueJ will ask you for a name for the new test method; let's call it **testAccountName**.

Once we have selected this name, we enter the *test recording* mode. An indicator on the left shows that we are now recording, and buttons are provided to end or cancel the recording. We

can now interactively carry out our test. Create a new **Student** object named James Duckling with student ID 123456. (You will also have to specify a birth date, but this does not matter much for our test.) Once the object appears

on the object bench, call its **getAccountName** method.

As before, we see the method result showing the (incorrect) account name, but when we're recording tests, there is an additional element in the result dialog box: an area to add an assertion. Here, we can now add the assertion for the expected result (Jam1234; see **Figure 7**).

Once done, close the dialog box and end the recording by clicking the **End** button under the recording indicator. BlueJ will immediately generate JUnit test code for this test and add it to the **StudentTest** class. You can open the editor to inspect it.

Playback

Once we have some unit tests recorded, we will naturally want to execute our tests at some point. There are three ways to do this:

- We can select an individual test method (for example, **testAccountName**) from the **StudentTest** class's pop-up menu to run only that test method.
- We can select **Test All** from the **StudentTest** class's pop-up menu to run all tests defined in that class.

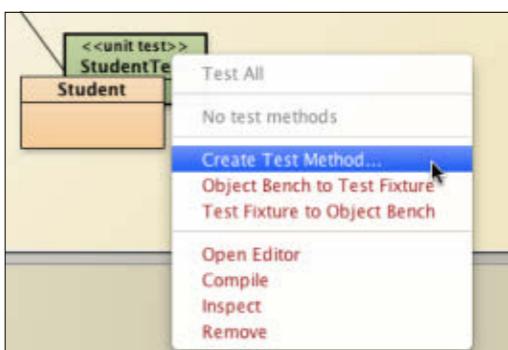


Figure 6

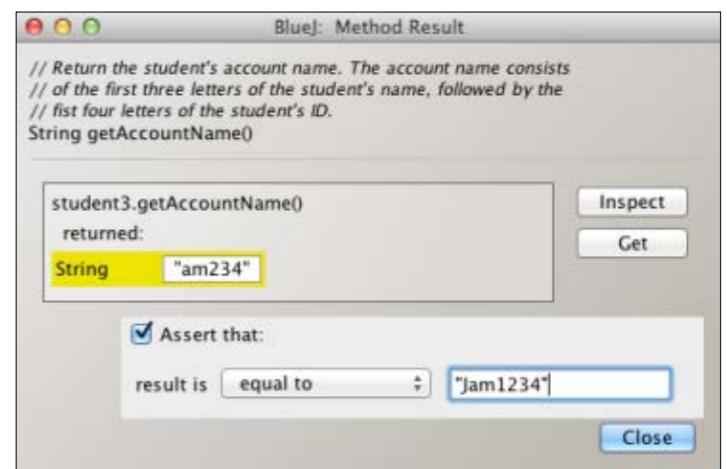


Figure 7

- We can use the **Run Tests** button in the left toolbar to run all tests in our project.

In each case, the results of the test runs are displayed (see **Figure 8**), and if an error was detected, BlueJ can take you straight to the JUnit source where the failure originated.

Test Fixtures

BlueJ can also use JUnit test fixtures. Just create your fixture objects interactively on the object bench as you would like them to be for the beginning of your

//new to java /

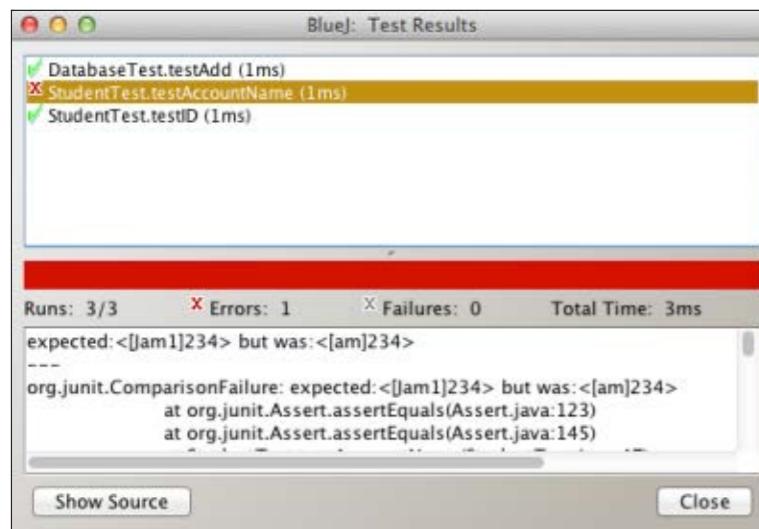


Figure 8

tests. Then use the **Object Bench to Test Fixture** function from the test class's pop-up menu, and the fixture will be added to your test class. At the start of each test recording, the fixture objects will be placed automatically onto the object bench for you.

Manual Test Editing

JUnit classes in BlueJ can also be edited manually, and you can freely mix manual writing of test code with adding recorded test cases. This allows interactive recording for many straightforward tests, while adding the ability to write more-complex tests by hand.

Over to You

You should now, of course, fix the bug we uncovered and make sure that all your tests pass. If you create some tests for the [Database](#)

class, you will discover that one of its methods is also faulty. Create some tests for each of its methods until you find one that fails, and then fix that one, too.

Conclusion

Testing is an important aspect for learning programming. The interactive testing feature in BlueJ usually leads to

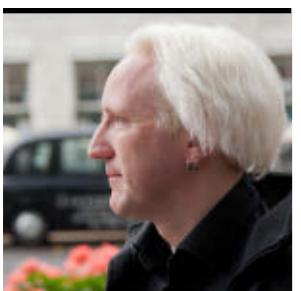
learners who test more often and earlier. The interaction also helps in constructing a good mental model of object interaction. Both of these aspects aid in the understanding of programming principles.

For more-systematic development, regression testing is important, and it should be taught in any good software development course. BlueJ's support for JUnit allows easy creation of unit tests for regression testing. The tight integration of interactive testing with regression testing provides probably the easiest mechanism available in any environment today for creating unit tests. </article>

[LEARN MORE](#)

- Java SE 7 API
 - BlueJ Website





Part 1

Exploring Lambda Expressions for the Java Language and the JVM



With lambda expressions, write code that is more concise and can truly run in parallel.



BEN EVANS,
MARTIJN VERBURG,
AND TRISHA GEE

BIO

Lambda expressions, a name chosen by the Expert Group to describe a new functional programming construct, are one of the most eagerly awaited new features coming in Java SE 8. You'll sometimes also hear people using terms such as *closures*, *function literals*, *anonymous functions*, and *Single Abstract Methods* (SAMs). Some of these terms have slightly different meanings from each other, but they all refer to the same basic functionality.

Although lambda expressions might seem unfamiliar to begin with, they're quite easy to pick up, and mastering them will be vital for writing applications that can take full advantage of modern multicore CPUs.

A key concept to remember is that lambda expressions are just small bits of func-

tionality that can be passed around as data. A secondary concept that you'll need to master is understanding how collections will be iterated over internally, as opposed to the external, serial iteration we have today.

In this article, we'll take you through the motivations behind lambda expressions, some use cases for them and, of course, the syntax.

Why You Want Lambda Expressions

There are three main reasons why programmers want lambda expressions:

- More-concise code
- Ability to modify methods by supplying additional functionality
- Better support for multi-core processing

More-concise code. Lambda expressions provide a neater way of implementing Java

classes that have only one function.

For example, if your code has a large number of anonymous inner classes—defining things such as listeners and handlers in UI code or callables and runnables in concurrent code—moving to the lambda expressions style can make your code much shorter and easier to understand.

Ability to modify methods. Sometimes, methods don't quite have the functionality we want. For example, the `contains()` method in the `Collection` interface returns `true` only if the exact object passed in is present in the collection. There's no way to tweak that functionality, for example, to have the method return `true` if the string we're looking for is present but uses different capitalization. Loosely, what we want

to do is "pass in some new code of our own" to an existing method, which will then call the code that we passed in. Lambda expressions can provide a good way to represent the code that has been passed in and should be called back.

Better support for multi-core processing. Today's CPUs have multiple cores. This means that your multi-threaded code can truly run in parallel, as opposed to being time-shared on a single CPU. Lambda expressions will help you write simple code that can use these cores efficiently by supporting the use of functional programming idioms in Java.

For example, you will be able to manipulate large collections in parallel using all the available hardware threads on your CPU by taking advantage of parallel



versions of idioms such as *filter*, *map*, and *reduce* (which we will be meeting shortly).

Lambda Expressions in a Nutshell

In the example of different capitalization of strings, what we want to do is pass a representation of `toLowerCase()`. In order to pass this code into a version of the `contains()` method as a second parameter, we'd need to do the following:

- Find some way to treat a code snippet as though it were a value (some sort of object)
- Find a way to be able to put that block of code into a variable

In other words, we want to wrap up bits of logic in something that can then be passed around. To make this a little more concrete, let's look at a couple of basic examples of lambda expressions that can be used to replace existing Java code.

Filtering. A good example of a snippet of code that you might want to pass around is a filter. For example, imagine you were using (a pre-Java SE 7) `java.io.FileFilter` in order to identify directories that belong inside a given path, as shown in **Listing 1**. With a lambda expression, this can be dramatically simplified, as shown in **Listing 2**.

The type (`FileFilter`) is inferred

from the left side of the assignment. The right side looks like a very shortened version of the `accept()` method in the `FileFilter` interface, that is, it accepts a `File` and returns the Boolean from evaluating `f.isDirectory()`.

In fact, we can actually simplify this further due to lambda expressions taking advantage of type inference, which works as follows. The compiler knows that `FileFilter` has only one method, `accept()`, so this must be the implementation code for that method. It also knows that `accept()` takes only one parameter, which is of type `File`. Therefore, `f` must be of type `File`. See **Listing 3**. As you can see, the use of a lambda expression radically reduces the amount of boilerplate.

Once you're used to lambda expressions, they make the flow of logic much easier to read. One of the key ways this is achieved is by locating the filtering logic next to the method that wants to use it.

Event handlers. Another area that features anonymous inner classes quite heavily is UI code. Let's take the assignment of a click listener to a button, as shown in **Listing 4**.

This is a lot of code just to say "when the button is pressed, call this method." Using lambda expressions, it's possible to use the code shown in **Listing 5**.

[LISTING 1](#) [LISTING 2](#) [LISTING 3](#) [LISTING 4](#) [LISTING 5](#) [LISTING 6](#)

```
File dir = new File("/an/interesting/location/");
FileFilter directoryFilter = new FileFilter() {
    public boolean accept(File file) {
        return file.isDirectory();
    }
};
File[] directories = dir.listFiles(directoryFilter);
```

[Download all listings in this issue as text](#)

The `listener` object can be reused if necessary, but if it's used only once, the code in **Listing 6** is considered good style.

In this example, the slightly odd syntax involving extra curly brackets is needed because `actionPerformed` returns `void`. We'll see more about that later.

Let's move on to take a look at the role that lambda expressions will play in writing modern code for handling collections, in particular the transition between two programming styles known as *external* and *internal iteration*.

External Versus Internal Iteration

Up until now, the standard way of dealing with a Java collection is via external iteration. It's called *external iteration* because there is control flow, external to the collection, which is used to iterate over the elements contained in

the collection. This is the traditional way of handling collections, with which most Java programmers are very familiar—even if they don't know or use the term.

For example, language constructs such as the enhanced `for` loop create an external iterator and use that object to traverse the collection shown in **Listing 7**.

With this approach, a collection class represents a "monolithic" view of all the elements in the collection, and the collection object can provide random access to any element of the collection that the programmer might require.

In this view of the world, we iterate by calling a method—`iterator()`—on the collection instance. This returns an object of type `Iterator`, which is a more restricted view on the same collection. It does not expose an interface allowing random access; instead, it is designed



for purely sequential handling of the contents of the collection. This sequential nature also gives rise to the infamous **ConcurrentModificationException** when you try to use the collection with parallel access.

The alternative is to require the collection object to manage the iterator (and loop) internally. This approach is called *internal iteration*, and it is the preferred choice when working with lambda expressions.

In addition to new syntax for lambda expressions, Project Lambda includes a major upgrade to the collections libraries. The aim of the collections upgrade is to make it much easier to write code that uses internal iteration to support a range of well-known functional programming idioms.

Functional Idioms with Lambdas

At one time or another, most developers have found themselves needing to perform one or more of the following operations on a collection:

- Create a new collection, which filters out any elements that don't satisfy some condition
- Apply a transformation to each element of a collection, and work with the transformed collection

- Build up an overall value for a property of the whole collection, for example, adding up or averaging the values

These tasks (which are called, respectively, the *filter*, *map*, and *reduce* idioms) have something important in common: They all require code to be run on each element of the collection in turn.

Whether the code is testing an element to see whether a condition is met (*filter*), transforming the element into a new element for a new collection (*map*), or computing a value to be fed into an overall calculation (*reduce*), the key theme is "a bit of code that needs to be run on each element."

This implies that we need a simple way of representing the code that is going to be used in internal iteration. Fortunately, Java SE 8 provides the building blocks for such representations.

Java SE 8 Classes for Basic Functional Idioms

Java SE 8 includes classes that are intended to be used to implement the basic functional idioms discussed above. These include **Predicate**, **Mapper**, and **Block**—as well as some others—in a new package called **java.util.functions**.

Let's look at **Predicate** in more detail. This class is often used to implement the filter idiom;

LISTING 7 **LISTING 8** **LISTING 9** **LISTING 10**

```
List<String> myStrings = getMyStrings();
for (String myString : myStrings) {
    if (myString.contains(possible))
        System.out.println(myString + " contains " + possible);
}
```

[Download all listings in this issue as text](#)

applying it to a collection returns a new collection containing only those elements that match the predicate.

There are a number of interpretations of what a predicate is. In Java SE 8, a predicate is a method that evaluates to true or false depending on the values of its variables.

Let's have another look at an example we saw earlier. We have a collection of strings that we want to test to see whether it contains a specific string, but we want the comparison to be done in a case-insensitive manner.

In Java SE 7, we would need to use external iteration, with code that looks like **Listing 8**. In the upcoming Java SE 8, we could write this in much more compact fashion using a **Predicate** and a new helper method (**filter**) in the Collections API, as shown in **Listing 9**. In fact, if you follow

common functional programming style, you'd write that in one line, as shown in **Listing 10**. As you can see, the code is still very readable and we have the benefit of using internal iteration as well.

Let's finish up by discussing the full syntax for lambda expressions in more detail.

Syntax Rules for Lambda Expressions

The basic form of a lambda expression starts with the list of parameters that can be accepted and ends with some code (called the *body*), with an arrow (**->**) separating the two.

Note: The syntax for lambda expressions is still subject to change, but at the time of this writing, the syntax shown in the following examples works.

Lambda expressions make fairly heavy use of type inference, which

is quite unusual compared to the rest of Java syntax.

Let's take a closer look at an example we've already seen (see Listing 11). If we look at the definition of `ActionListener`, we can see it's an interface with just one method in it (see Listing 12).

The lambda expression on the right side of Listing 11 can, therefore, easily be understood to mean "here is the method definition for the single method declared in the interface." Notice that the usual rules of Java static typing must still be obeyed; this is the only way that type inference can work correctly.

From this, we can see that lambda expressions can be used as compact ways of writing code that would previously have used anonymous inner classes.

There's one other syntax quirk that you should be aware of. Recall the example shown in Listing 13. At first glance, this looks just like the `ActionListener` example. However, let's take a look at the definition of the `FileFilter` interface (see Listing 14).

The `accept()` method returns a Boolean value, but there's no explicit `return` statement. Instead,

[LISTING 11](#) [LISTING 12](#) [LISTING 13](#) [LISTING 14](#)

```
ActionListener listener = event -> {ui.showSomething();};
```

[Download all listings in this issue as text](#)

the return value is inferred from the expression—in this case, `f.isDirectory()`—which is, of course, a Boolean value.

This explains why methods that return `void` need to be treated a bit differently. The extra syntax is used to put an additional set of braces around the code part of the lambda expression. Without this syntax quirk, type inference would not work properly—but you should be aware that this syntax might change.

A lambda expression's body can contain multiple statements. In such a case, the body should be surrounded by braces, the "inferred return" syntax does not work, and the `return` keyword must always be present.

One final word of warning: your IDE is unlikely to support the syntax at this stage, so you'll need to pay close attention to any compiler

warnings that `javac` produces when you first try lambda expressions.

Conclusion

Lambda expressions are the largest new language feature in Java since generics were introduced in Java SE 5. Done right, lambda expressions will allow you to write code that is more concise, modify methods by supplying additional functionality, and make multicore processors sing. By now, we're sure you must be eager to try lambda expressions, so without further ado . . .

You can get a snapshot build of Java SE 8 with lambda expressions at the official [Project Lambda](#) page. As well as trying out the binaries, you should also read the "State of the Lambda" articles hosted there. </article>

LEARN MORE

- For help with lambda expressions or to join our global hack days, join the [Adopt OpenJDK project](#)



FIND YOUR JUG HERE

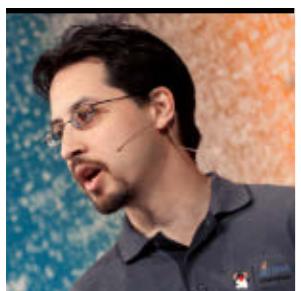
My local and global JUGs are great places to network both for knowledge and work. My global JUG introduces me to Java developers all over the world.

Régina ten Bruggencate
JDuchess

[LEARN MORE](#)



ORACLE®



Part 1

JavaFX in Spring

Why use the Spring framework on the client?

STEPHEN CHIN

BIO

If you have done any server-side Java development, you have probably used some part of the Spring framework to build or augment your Java enterprise Web applications. It comes with a powerful dependency injection (DI) system, Web model-view-controller (MVC) framework, and helper libraries for everything from database access to security authorization control.

With the 2.x release, JavaFX now has pure Java APIs, which makes integrating Java libraries, such as Spring, trivially easy. Some of the benefits you can get by taking advantage of the Spring framework on the client include the following:

- Inversion of UI Control—Inversion of Control (IoC) is an object-oriented technique for decoupling the execution of a task from its implementation. For UIs, this is very useful for creating standalone mod-

ules that have their relationships injected using a dependency management framework such as Spring.

- One-line Web services—By using a compatible REST framework on the client and server, you can reuse the same model objects and implement each Web service call with a single statement for retrieval, creation, deletion, and even updates.
- Authentication and authorization—Rather than reinventing the wheel on the client, you can take advantage of the same authentication mechanism you use to secure your server requests. This also lets you do fine-grained authorization tests to modify the UI experience based on the permissions. For very simple applications, these benefits may not outweigh the costs of learning and integrating a new library.

However, if you are building a large, multiscreen application that connects to a back-end server or authenticates users, you will get an exponential benefit from these features as the size and complexity of your application grows.

To demonstrate these benefits in the context of a real application, we will build out a customer application from scratch with both a front-end

and a back-end component done as a hybrid Spring Framework and JavaFX application. It will include an authentication module to log in users, back-end Web service integration using the Spring [RestTemplate](#), and create and delete operations that are appropriately permissioned by role. The final application will look like **Figure 1**.



Figure 1



//rich client /

While this example will be built out over the course of a two-part series, you can browse, download, and run the full example from the [source code at GitHub](#) right now.

An Application Template

If you have done any Spring development on the server side, you know that configuring a new application to use Spring is as simple as adding in the JAR files (or Maven dependencies) and adding a couple of lines to your web.xml file.

On the client, you need to take care of manually starting the Spring runtime from your main class. The application template we create in this section will show you how to start the JavaFX and Spring runtimes in a thread-safe manner.

But first, let's start with a little background on JavaFX threading to explain why this matters.

JavaFX has two different threads, each with a distinct purpose:

- Application thread—This is the thread where all manipulation of the JavaFX scene graph and the creation of objects that use top-level windows should occur. It is analogous to the Swing event dispatch thread (although that is not the same, so be careful with Swing interoperability code).
- Render thread—The render thread is what gets called on

each graphics pulse to prepare the scene for display. This includes processing animation, CSS, and layouts; synchronizing the scene graph with the render tree; and updating the mouse hover state.

For most purposes, the render thread is something behind the scenes that you don't have to worry about; however, the application thread does require a little bit of attention.

Whenever JavaFX calls you back either in application methods such as `start` and `stop` or in an event handler, it will do so on the application thread. This allows you to safely create new JavaFX objects and manipulate the scene graph freely. However, if you are on the main thread or you spawn your own thread, be careful not to call any JavaFX methods that would affect the render tree or you will get an exception, deadlocks, or worse.

Listing 1 ([CustomerModel.java](#)) shows the main application class of a `CustomerApp` demo project that takes care of initializing the JavaFX application threads as well as loading the Spring configuration from a Java class.

This code is very similar to a boilerplate JavaFX application until you get to the `start` method. This is where you initialize the Spring `ApplicationContext` by creating a

LISTING 1

```
public class CustomerApp extends Application {
    public static void main(String[] args) {
        launch(args);
    }

    @Override
    public void start(Stage stage) throws Exception {
        ApplicationContext context =
            new AnnotationConfigApplicationContext(
                CustomerAppConfiguration.class);
        ScreensConfiguration screens = context.
            getBean(ScreensConfiguration.class);
        screens.setPrimaryStage(stage);
        screens.loginDialog().show();
    }
}
```

[Download all listings in this issue as text](#)

new `AnnotationConfigApplicationContext` if you are using a Java configuration or a `ClassPathXmlApplicationContext` if you are loading from XML.

I highly recommend using the new annotation-based Java configuration support that was introduced in Spring Framework 3.0, because it provides all the convenience of defining your configuration in the code without sacrificing the power and debugability that the Spring XML-based configuration provides. In this example, we will actually mix both styles of declaring configuration, which can be conve-

nient depending on what you are doing with your application.

While it might be tempting to initialize the Spring context in the main method of your application, this is a very dangerous practice because loading the Spring context might cause JavaFX objects to get loaded on the thread from which you started Spring. If you happen to create a JavaFX object that uses top-level windows, such as Stage, Popup, Menu, and so on, you will get a nice exception to remind you of this fact. The `start` method, however, is called on the JavaFX application thread, which

//rich client /

means that you can safely create JavaFX objects of any type and add them to the scene graph even from within your Spring configuration.

In this particular example, we load a subconfiguration class called `ScreensConfiguration`, set the stage on that configuration object, and call a method to show the login dialog box.

Modularizing Your UI With DI

Now that you are able to load Spring, it is time to take advantage of some of its features to help modularize your JavaFX application. The finished Customer Data App will include the following screens:

- A login screen
- A data screen
- An error screen
- An “add customer” screen

Each of these screens will be implemented as a separate JavaFX class or FXML file, and they will require references to other screens (when navigation is required) and a model (when data access or updates occur). We also want the screens to get created on first use, but we want to have no more than a single instance of each screen at a given time.

Listings 2a and 2b

(`ScreensConfiguration.java`) show a Spring configuration class that defines each of our screens as a separate bean.

In the `ScreensConfiguration` file, we define each screen and associated controller as a separate bean by using the `@Bean` annotation. Also, some of the beans (in our case, all that display dialog boxes) are defined with `@Scope("prototype")` so that a new instance gets created each time we fetch one. However, the main data screen gets the default scope of `"singleton"`, which means we will have only one main screen for the entire application no matter how many times the method is called.

Note that the entire configuration class is annotated with `@Lazy`, to ensure that beans will not get created until the first time they are accessed. However, this alone is not enough to ensure that beans do not get created on startup, because direct links between screens will cause a domino effect where loading the first screen loads every other referenced screen.

To avoid loading all the screens at once, rather than injecting a reference to the screens we want to connect, we instead pass in a reference to the `ScreensConfiguration` class. This still gives us the flexibility to swap implementations by providing a different `ScreensConfiguration` instance, but it defers object construction until the method for loading a specific screen gets called.

LISTING 2a LISTING 2b

```
@Configuration
@Lazy
public class ScreensConfiguration {
    private Stage primaryStage;

    public void setPrimaryStage(Stage primaryStage) {
        this.primaryStage = primaryStage;
    }

    public void showScreen(Parent screen) {
        primaryStage.setScene(new Scene(screen, 777, 500));
        primaryStage.show();
    }

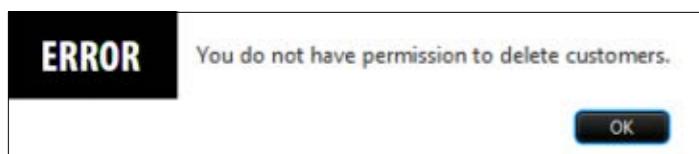
    @Bean
    CustomerDataScreen customerDataScreen() {
        return new CustomerDataScreen(customerDataScreenController());
    }

    @Bean
    CustomerDataScreenController customerDataScreenController() {
        return new CustomerDataScreenController(this);
    }

    @Bean
    @Scope("prototype")
    FXMLDialog errorDialog() {
        return new FXMLDialog(errorController(),
            getClass().getResource("Error.fxml"), primaryStage,
            StageStyle.UNDECORATED);
    }
}
```



[Download all listings in this issue as text](#)

**Figure 2**

The rest of the configuration for the model and Web services is defined in a separate class file that loads the [ScreensConfiguration](#) via annotation, as shown in [Listing 3](#) ([CustomerAppConfiguration.java](#)).

Injecting Controllers into FXML

In the previous section, we defined several screens, some as Java class files and others as FXML. While it is fairly straightforward to attach a Spring-initialized controller to a Java object, doing the same for FXML was not even possible with the JavaFX 2.0 release. Fortunately, the JavaFX team added an API in the 2.1 release to specifically address this for IoC frameworks such as Contexts and Dependency Injection (CDI), Guice, and, of course, Spring.

As an example, let's take a closer look at the definition of the [ErrorDialog](#). The view and controller are both created in the [ScreensConfiguration](#) class with the lines shown in [Listing 4](#).

Notice that the [errorController](#) is injected as a dependency into the [FXMLDialog](#) class. This is a custom class I put together to encapsu-

late the logic for loading an FXML file as well as injecting the controller object. The full source code for the [FXMLDialog](#) is shown in [Listing 5](#) ([FXMLDialog.java](#)).

This class generalizes some of the code you would normally need in order to load an FXML file as a new dialog (Stage). However, the important part for injecting the controller is the call to [loader.setControllerFactory\(\)](#), which creates an inner class that will return the controller passed in as a parameter.

By using this technique to create and inject the controller, it is treated the same as any other Spring-managed object, letting you use DI, autowiring, and other features of the framework. The full code for the [ErrorController](#) is shown in [Listing 6](#) ([ErrorController.java](#)). And the base [DialogController](#) interface is shown in [Listing 7](#) ([DialogController.java](#)).

The FXML UI was created using [SceneBuilder](#), which is a great visual tool for learning the JavaFX components and quickly mocking up UIs. The finished error dialog box I created looks like [Figure 2](#).

Conclusion

In this article, I have given you an application template for using Spring and JavaFX together that

[LISTING 3](#) [LISTING 4](#) [LISTING 5](#) [LISTING 6](#) [LISTING 7](#)

```
@Configuration
@Import(ScreensConfiguration.class)
@ImportResource("classpath:applicationContext-security.xml")
public class CustomerAppConfiguration {

    @Bean
    CustomerModel customerModel() throws IOException {
        CustomerModel customerModel = new CustomerModel();
        customerModel.setRestTemplate(restTemplate());
        customerModel.loadData();
        return customerModel;
    }

    @Bean
    RestTemplate restTemplate() {
        RestTemplate restTemplate = new RestTemplate();
        restTemplate.setMessageConverters(
            Collections.<HttpMessageConverter<?>>singletonList(
                new MappingJacksonHttpMessageConverter()));
        return restTemplate;
    }
}
```

 [Download all listings in this issue as text](#)

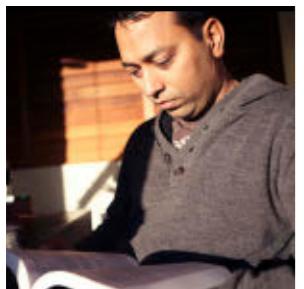
you can reuse in your own projects. Just taking advantage of Spring configuration and DI already gives your application design an advantage over writing in pure Java.

In Part 2, I will build out the rest of the application including

a lightweight back end with reusable model objects and one-line Web services on the client using [RestTemplate](#). </article>

LEARN MORE

- [Stephen Chin's blog](#)



Part 2

Wirelessly Recover Your Device's Address Book

VIKRAM GOYAL

BIO 

Learn how to add a recovery mode to the backup capability of your device's address book.

In Part 1, I created an application that allowed you to back up your device's address book to a remote destination specified as a Bluetooth destination. The application had several functions: discovering destinations, setting a schedule, performing on-demand backups, and viewing logs. However, it lacked a recovery mode. In this follow-up article, I will add that missing element.

Note: The source code for the application described in this article can be downloaded as a [NetBeans project](#) and as a [desktop application](#).

The Application Flow

Figure 1 shows the target application in action with the new option added (and selected).

Unlike the backup functionality in the previous

application, which you could schedule to start automatically, restoring must be an on-demand service. Hence, I have provided it as an explicit option. In a real-life scenario, if the user lost his address book data (either because he got a new phone or there was some sort of data corruption), he would try to recover it using this new application (provided a backup had been done).

To do the restore, we have to turn the backup process on its head. The steps involved in a backup were

1. Iterate over the individual address book items.
2. Compress them into an [LZCOutputStream](#).
3. Send the data to a pre-determined Bluetooth destination.

In the reverse recovery process, the steps are as follows:

1. Receive the data (that is, the backup file) from the Bluetooth destination.
2. Decompress the data.
3. Add the individual items to the user's address book.

I will follow these steps in creating the restore functionality.

Receiving the Backup File via Bluetooth

To receive the data.gz backup file that we created in the previous article, the user has to initiate the application on both the backup device and the phone. The application will then actively listen for any incoming data requests.

To implement this, I have created a new method called

[doReceive\(\)](#) (see **Listing 2**) in the [BluetoothServices](#) class (from the existing code). This method is called by the [startRestore\(\)](#) method (see **Listing 1**) in the main [MIDlet](#) class and is the only call in the [startRestore\(\)](#) method.

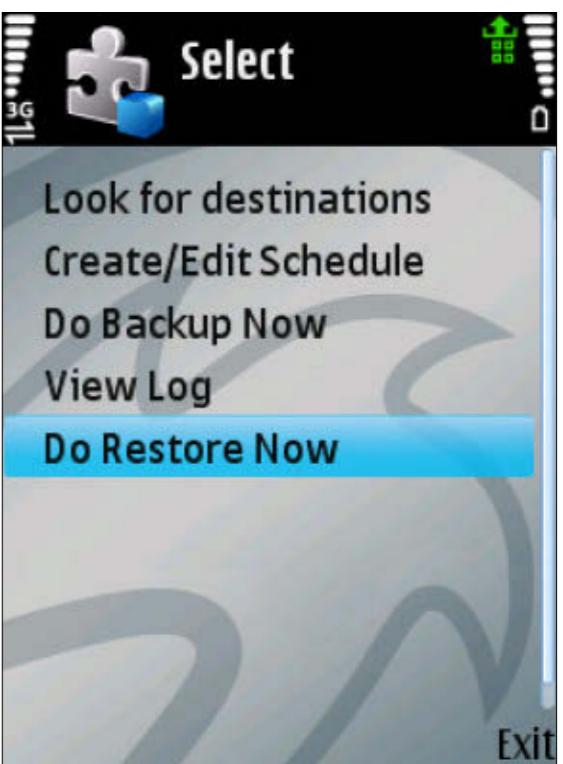


Figure 1

//mobile and embedded /

Listing 2, which creates a local server on the phone, is used to create this listening socket in a separate thread. The identification of this server is done by using the name “Backup” and then creating a **SessionNotifier** that will accept new connections. Check out **Figure 2** to see how this process looks when the user initiates it on the phone.

Notice that we have set a [RequestHandler](#) object to listen for new connections as they are made and to handle them. It is this class, shown in [Listing 3](#), that accepts the connections from the remote backup computer and makes them available to the calling MIDlet. This [RequestHandler](#) extends the

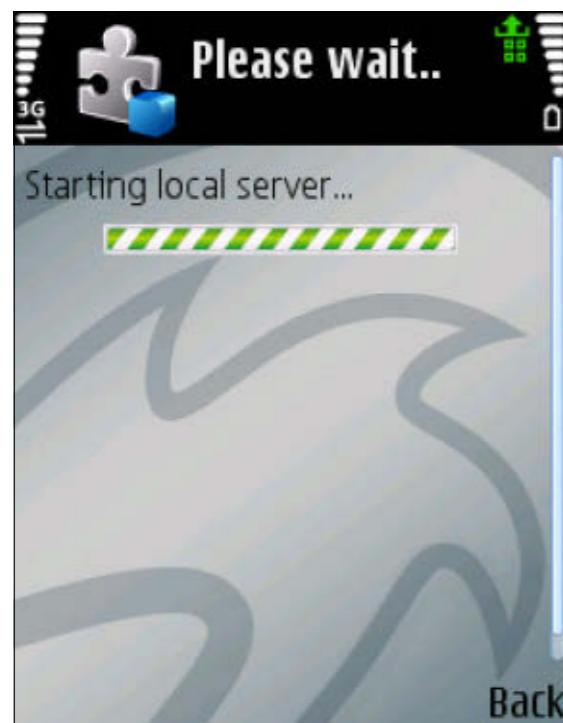


Figure 2

The `ServerRequestHandler` class from the `javax.obex` package. I have handled only the methods that are essential for the completion of this exercise.

The real point of interest in this code is when the `onPut()` method is called. When the server thread on the phone receives a connection, this method handles the interaction. It receives the backup file (as a .gz compressed file) from the remote (backup) computer and then gives the file to the callback method in the originating MIDlet.

The question now arises as to the mechanism for sending this file from the backup computer. If you were to use the built-in Bluetooth software on the backup computer, it would send the backup file only to the phone's default mechanism for handling such files, *not* to your application. In most phones, the default mechanism for receiving files is to receive them as SMS messages. This is not ideal, because our MIDlet application has no way to retrieve the data.gz backup file.

It is clear that we need to send this file using our own software interface. For this, I have created a simple desktop-based class, which essentially is the code in the `doSend()` part of the `BluetoothServices` class. This code

LISTING 1 / LISTING 2 / LISTING 3

```
private void startRestore() {  
    log("Restore Started At: " + new Date());  
    btServices.doReceive(this);  
}
```

 Download all listings in this issue as text

//mobile and embedded /

is shown in **Listings 4a** and **4b** and uses the [BlueCove library](#), which is a JSR 82 implementation.

Try as I might, I wasn't able to get the right service code to identify my phone. The service was discovered but the name wasn't always the same as what I was expecting, so I had to hack this by identifying the correct service by the presence of the number 5. This number might be different for you.

To run this code on your desktop, you will need to have the BlueCove directory in your class path and the data.gz file must be present in the same directory where you are running this code. The combination of fulfilling these two requirements

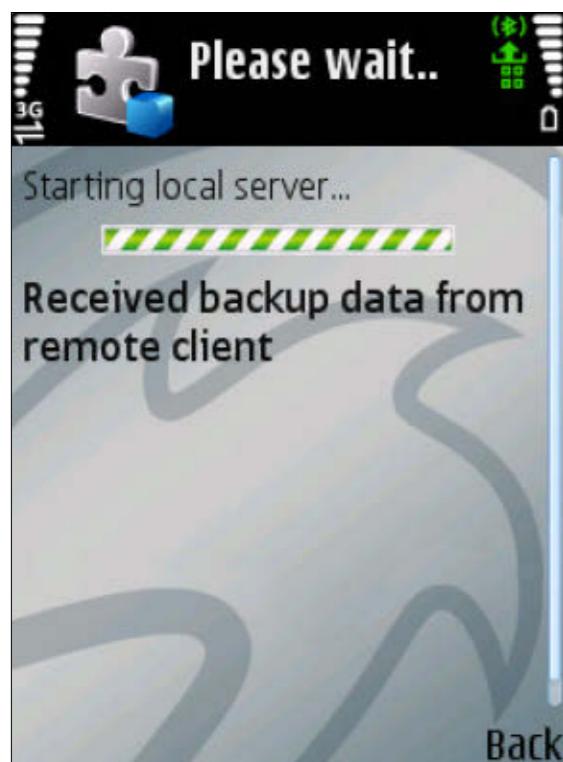


Figure 3

in parallel allows the data.gz file to be sent to the phone (see **Figure 3**).

Decompressing the Data

Now that we have received the data from the remote backup computer, the rest of the steps are easier.

In order to decompress the data, we simply pass it through the LZC decompressor. The `receiveBackup(byte[] data)` method in the MIDlet application is called by the `RequestHandler` method once the data has been successfully received. The first step in this method is the decompression of the data:

```
LZCInputStream lzc =
    new LZCInputStream(
    new ByteArrayInputStream(data));
```

Adding the Individual Items to the User's Address Book

The Personal Information Management (PIM) API helps us in this step, although only to an extent. It provides the `PIM.getInstance().fromSerialFormat()` method (see **Listing 5**), which reads an input stream to decipher the contents and then automatically creates the right PIM item from the contents.

This is where the usefulness of this method ends. Unfortunately, it reads only the first entry in the

LISTING 4a

LISTING 4b

LISTING 5

```
public void servicesDiscovered(int transID,
ServiceRecord[] servRecord) {
    Operation op = null;
    OutputStream out = null;
    for (int i = 0; i < servRecord.length; i++) {
        String connURL = servRecord[i].getConnectionURL(
            ServiceRecord.NOAUTHENTICATE_NOENCRYPT, false);
        System.out.println(connURL);
        // found the right service?
        if (connURL.contains("5")) {
            System.out.println("Found service, connecting to it...");
            try {
                // read the local backup file
                InputStream input =
                    new BufferedInputStream(new FileInputStream("data.gz"));
                // read it in to our bucket
                byte[] bucket = new byte[32 * 1024];
                ByteArrayOutputStream result = null;
                result = new ByteArrayOutputStream(bucket.length);
                int bytesRead = 0;
                while (bytesRead != -1) {
                    bytesRead = input.read(bucket);
                    if (bytesRead > 0) {
                        result.write(bucket, 0, bytesRead);
                    }
                }
            }
```



[Download all listings in this issue as text](#)

file; as soon as it encounters the end of this entry (by means of the END vCARD), it stops. It leaves the `InputStream` object open and leaves its character position on the character after the end of the entry (if there are no errors). Thus, this method allows you to read only a

single entry, which is not ideal if you are trying to recover multiple entries from the backup file.

It is easy to overcome this, however, because the method hints that you can iterate over the `InputStream` to get the rest of the entries. I will leave this as an

//mobile and embedded /

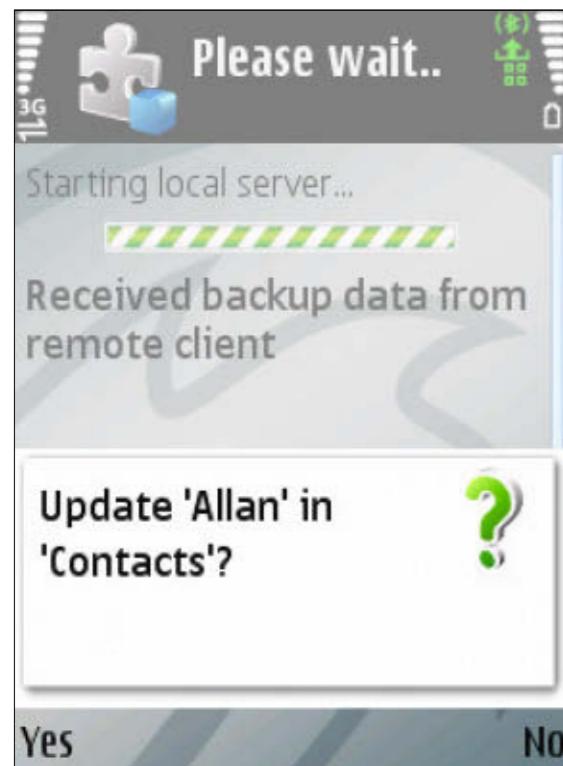


Figure 4

exercise for the reader.

Figure 4 and **Figure 5** show this functionality running on the phone. We have successfully restored our contacts!

Conclusion

This was the final part of a two-part series on identifying how we could back up and restore our address book in a Java ME-enabled phone over the network. In this part, we looked at how we could create a server on the phone to listen for incoming connections over the Bluetooth network and then use that server to receive the backup file and restore the contacts into our address book.

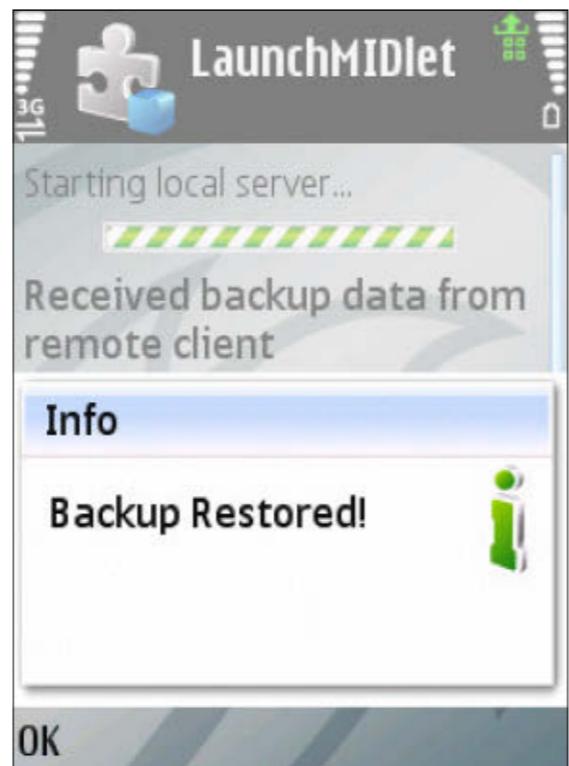


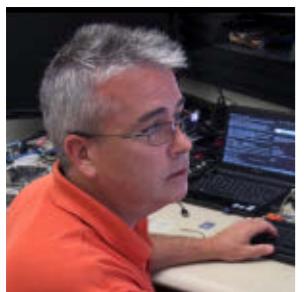
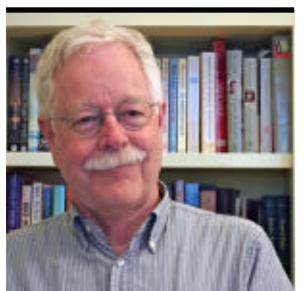
Figure 5

To be able to use this application, we had to create a separate desktop application that did the heavy work on the backup computer, bypassing the built-in mechanism for sending files via SMS. This enabled our phone's MIDlet application to "catch" the file and process it. </article>

[LEARN MORE](#)

- Bluetooth article on discovering and sending data
 - The compress-j2me project
 - BlueCove project, a Java library for Bluetooth for interaction on the desktop





BILL COURINGTON AND GARY COLLINS

BIO

Get Started with Java SE for Embedded Devices on Raspberry Pi

How to get Linux and Java SE for Embedded Devices running on the Raspberry Pi in less than an hour.

It is hardly larger than a credit card. It costs about the same as a book on Java programming.

It's the Raspberry Pi computer, and it can support a full Java SE runtime for headless embedded applications. This article gets you started with Java on the Raspberry Pi.

Note: There are many potential variations of the instruc-

tions given here, especially with respect to which operations you execute on a host (desktop or laptop) computer and which you execute on the Raspberry Pi. Use your knowledge and preferences (and perhaps some of the topics in the “Performing Optional Linux Tuning and Tweaking” section of this article) to create a workflow that works for you.

The Raspberry Pi

The Raspberry Pi is a small, low-power board built around a 700 MHz ARMv6 CPU with hardware floating point and a graphics processor integrated in a single chip. The graphics processor and the CPU share 256 MB of RAM. There are connections for USB, Ethernet, high-definition graphics, audio, an SD card, and general-purpose I/O.

Figure 1 shows a Raspberry Pi model B with an SD card inserted for scale. The SD card plays a special role: the Raspberry Pi boots from it. This article largely consists of instructions for creating and modifying files on a bootable SD card.

You can interact with the Raspberry Pi using a USB keyboard and mouse and an HDMI monitor or television (up to 1,080 p). The board's

Ethernet connection gives another option: interacting from a networked host via [ssh](#). The “Performing Optional Linux Tuning and Tweaking” section of this article describes how to enable [ssh](#) logins.

Prerequisites

To set up the Raspberry Pi model B to run Java SE for Embedded Devices, you need

- A wired network to which you can connect the Raspberry Pi by an Ethernet cable. The network must supply DHCP IP addresses.
 - A Linux host computer on the same wired Ethernet network as the Raspberry Pi. You can perform equivalent operations on a Microsoft Windows or Apple Mac host, but the details differ and this

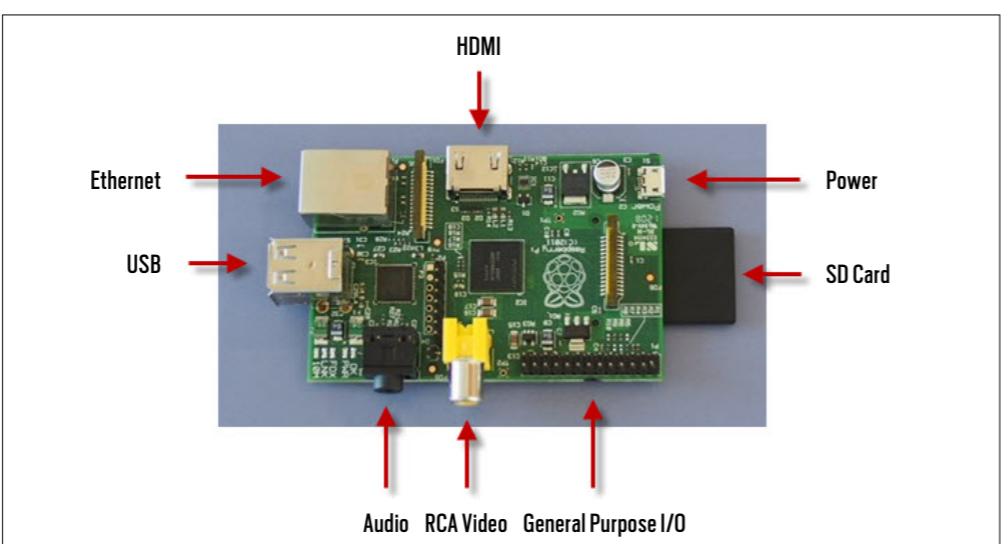


Figure 1

//mobile and embedded /

article does not describe them. You need the root password for the host computer. On the host you also need the following:

- A way to download a file from a Website, such as a Web browser or the `wget` utility.
 - A disk partitioning tool. We use GParted here, which is simple and effective. If you don't have it, you can acquire and install it on the host as follows:

sudo apt-get install gparted

- A Web browser and e-mail client for downloading Java SE for Embedded Devices from Oracle.
 - An SD card reader/writer. If your host is running Linux in a virtual machine, be sure it can read from and write to SD cards.

A monitor that can connect directly or through an adapter to the Raspberry Pi's HDMI port.

A USB keyboard and mouse connected to the Raspberry Pi by a powered USB hub.

An SD card to hold the Raspberry Pi's operating system and Java runtime. A 4 GB card has adequate capacity for Linux, Java SE for Embedded Devices, and most embedded applications.

Note: Not all SD cards work with the Raspberry Pi. High-

speed cards can be too fast for the Raspberry Pi bus. We have been successful with Transcend 4 GB Micro and Patriot 4 GB Class 4 cards. The [RPi VerifiedPeripherals](#) page has lists of cards that have and have not worked for Raspberry Pi users.

- A 5 V DC micro-USB power supply capable of at least 700 mA. As mentioned on the Raspberry Pi Website, do *not* use a USB hub or a computer as a power source.

Performing the Essential Linux Setup

When power is applied, the Raspberry Pi firmware boots from the SD card slot. Java SE for Embedded Devices runs on Linux; therefore, your first task is to get Linux on an SD card.

Download the Debian Squeeze Linux image for the Raspberry Pi.

Note: These instructions are for Debian Squeeze. Other Linux implementations for the Raspberry Pi might work, but they *must* have been built for the ARM chip's software floating point capability. Images referenced by the Raspberry Pi Website might have been built for hardware floating point capability; they will *not* work with Java SE for Embedded Devices.

LISTING 1

```
[ 110.084625] sdb: detected capacity change from 0 to  
3963617280  
[ 118.055249] sd 2:0:0:0: [sdb] 7741440 512-byte logical  
blocks: (3.96 GB/3.69 GiB)  
[ 118.059409] sd 2:0:0:0: [sdb] Assuming drive cache:  
write through  
[ 118.064547] sd 2:0:0:0: [sdb] Assuming drive cache:  
write through  
[ 118.066015] sdb: sdb1
```



[Download all listings in this issue as text](#)

1. Download the Debian Squeeze Linux image to a new directory called ~/RaspberryPi. You can find a link to a zipped Debian Squeeze image for the Raspberry Pi [here](#). The file size is about 450 MB. To verify the downloaded file, follow the SHA-1 Checksum instructions on the page that is displayed when the download starts. The following instructions assume you have downloaded the image to a new directory ~/RaspberryPi.
 2. Unzip the image as follows:

```
| $ cd ~/RaspberryPi  
$ unzip *.zip
```

Unzipping creates a subdirectory called something like **debian6-19-04-2012**.

Copy the Debian Squeeze image to the SD card.

- 1.** On your host computer, discover the SD card device handle as follows (or use a different way if you prefer):
 - a.** In a Linux terminal window, run `dmesg | tail`, which shows messages associated with device mounts. Only the final messages are of interest.
 - b.** Insert the SD card into the reader/writer.
 - c.** In a second terminal window, run `dmesg | tail` again. The additional lines in the output (compared to the first window) relate to the newly mounted SD card. They will vary in detail but should look something like the output shown in **Listing 1** (some lines have been broken to fit the available space).

//mobile and embedded /

5. In the dialog box that appears, drag the right arrow until **Free space following (MiB)** is about 512 (or however much you want to leave for

swap), as shown in **Figure 4**. You might not be able to drag to exactly 512, but the exact size is not important.

6. Click **Resize/Move** to queue

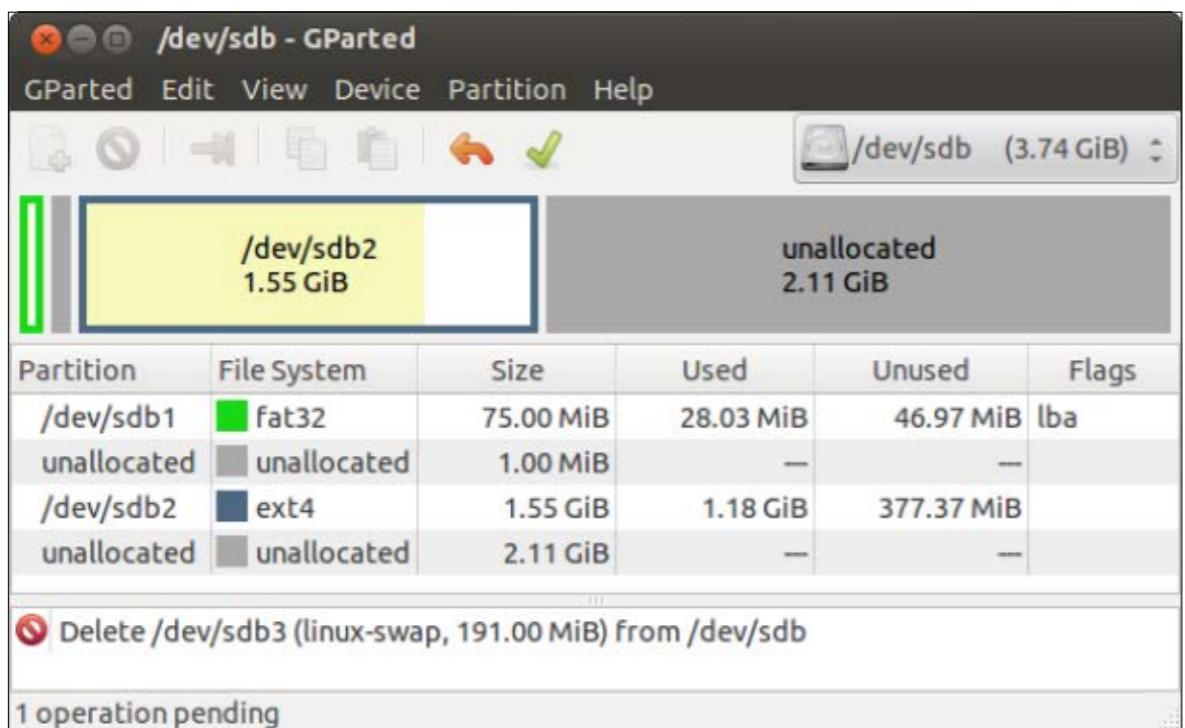


Figure 3

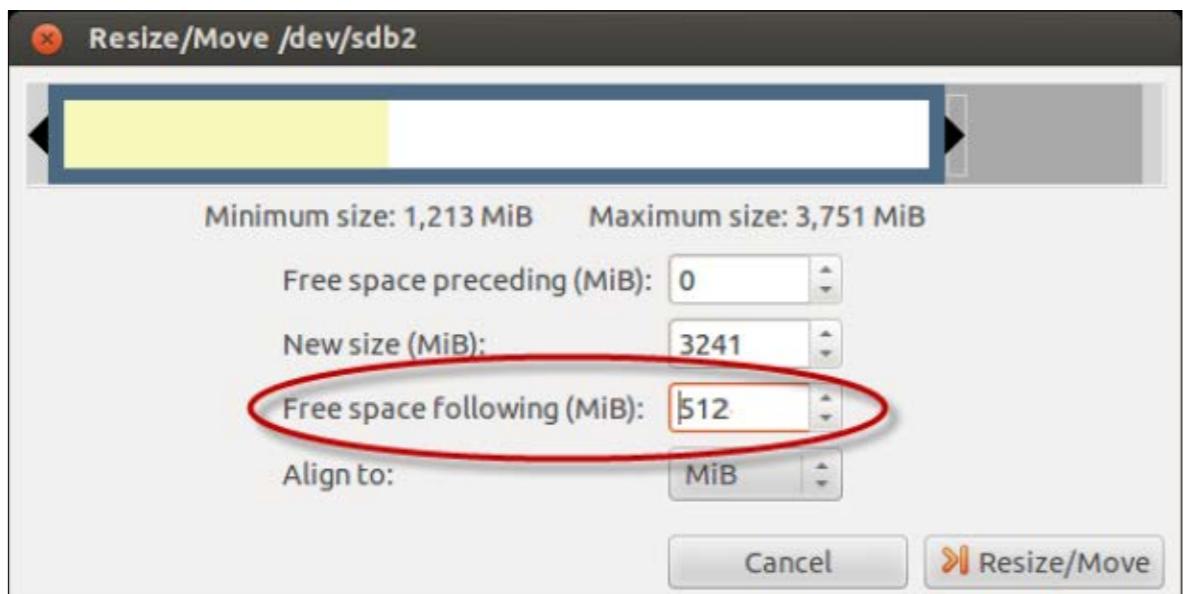


Figure 4

- the change.
7. In the “Moving a partition” warning dialog box that appears, click **OK**.

Now that we’ve grown the file

system partition, we re-create a swap partition.

8. Select the 512 MiB unallocated partition, right-click, and choose **New**.

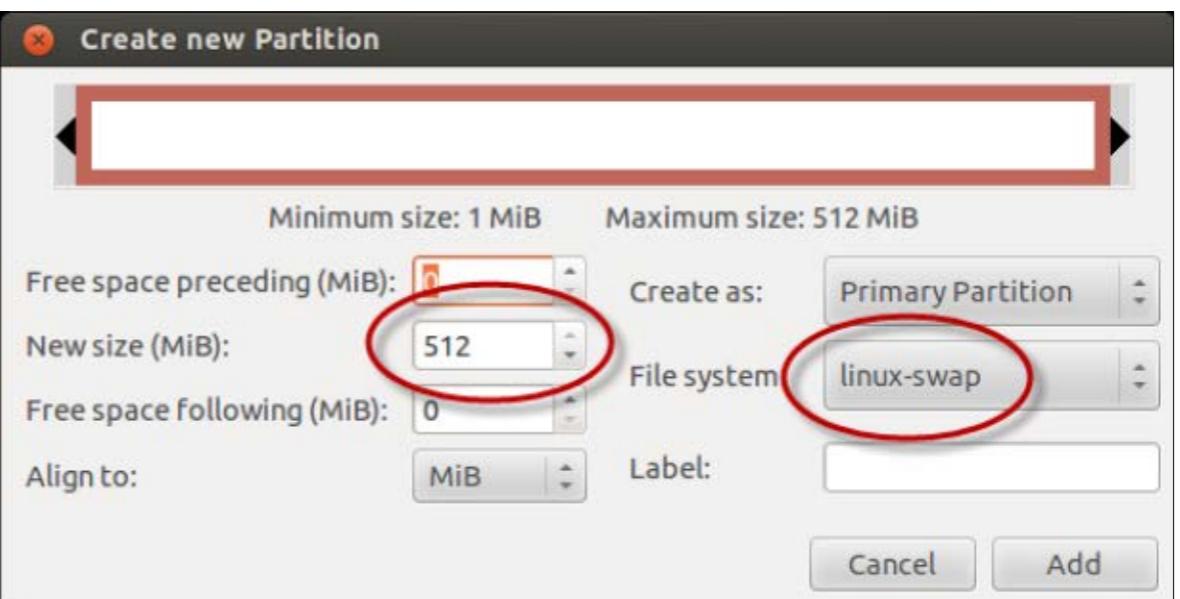


Figure 5

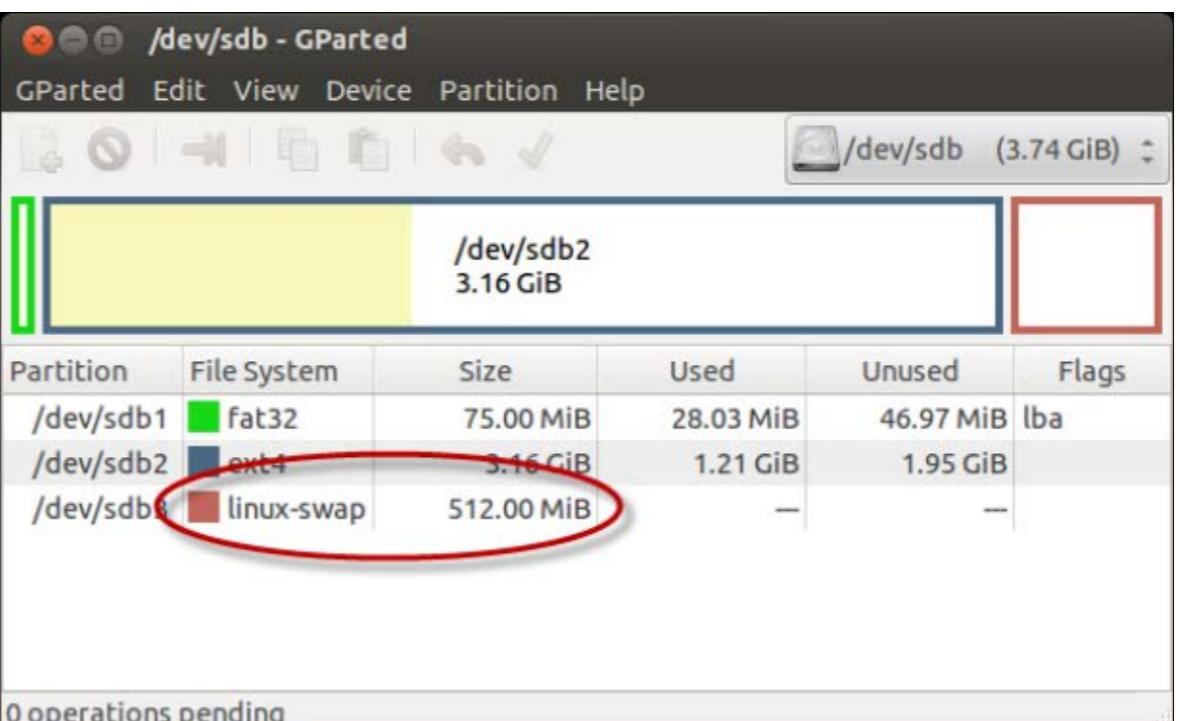


Figure 6



//mobile and embedded /

name, and aliases.

Listing 4 shows a hypothetical example in which we designate the Raspberry Pi as `raspberrypi`.

3. Save the file and exit the editor.
4. As superuser, open `/etc/hostname` in a text editor.
5. Add a line analogous to this: `raspberrypi`.
6. Save the file and exit the editor.

The following instructions can prevent a problem in which an overloaded network causes the operating system to change the Raspberry Pi's static IP address. As a result, `ssh` or other operations that use the static IP address stop working.

1. Discover the Ethernet controller's hardware address by running the command shown in **Listing 5** (some output lines have been broken to fit the text column).

In **Listing 5**, each `x` in the output stands for an address digit. In this example, the name of the Ethernet controller is `eth0`. The hardware address consists of the hex digits following `link/ether`, so in this example, the address is `b8:27:eb:b5:e8:90`.

2. Now that you have the Ethernet controller's hard-

ware address, as superuser, open `/etc/network/interfaces` in a text editor and add lines analogous to those that follow the `# New entries` comment shown in **Listing 6** (substitute your network details).

Enable swapping and optimize file system access time. By default, swapping is disabled in Debian Linux. If an operation exceeds the Raspberry Pi's 256 KB RAM, the system crashes. Less serious is the fact that, by default, the file system spends time maintaining a last-accessed time for each file, which is of little use in embedded applications. Maintaining last-accessed times can also reduce the life of an SD card. Both default behaviors are specified in the `/etc/fstab` file.

To enable swapping, do the following.

Note: Swapping is slow on an SD card. For an embedded application, you probably want to manage memory so as to avoid swapping.

1. As superuser, open `/etc/fstab` in an editor such as vi. The file looks like **Listing 7**.
2. To enable swapping, uncomment the third line.
3. To eliminate the last-accessed overhead, insert a `p2` mount point between `p1` and `p3`. The `noatime` and `nodiratime` options do the job.

LISTING 4 **LISTING 5** **LISTING 6** **LISTING 7** **LISTING 8**

```
127.0.0.1 localhost 192.168.0.100 raspberrypi.yourDomain
raspberrypi loghost
```

[Download all listings in this issue as text](#)

When the file looks similar to **Listing 8**, save it and exit the editor.

4. Now assign `p3` (the swap partition we created on the host as `/dev/sdb3`) to the swap space:

```
$ sudo mkswap /dev/
mmcblk0p3
```

Set the time zone and locale. By default, the Raspberry Pi is configured for the Europe/London time zone and the `en_GB.UTF-8` locale. To localize your computer, follow these instructions.

1. To change the time zone, enter this command:

4. Now assign `p3` (the swap partition we created on the host as `/dev/sdb3`) to the swap space:

5. To set the time zone, run the command `sudo dpkg-reconfigure tzdata`. A pseudographical user interface appears, as shown in **Figure 9**.
2. Move the selection cursor through regions and cities using the Up and Down arrow keys. Use the Left and Right arrow keys to highlight `<Ok>` or `<Cancel>` and press `Return`.
3. Set the locale by running the following command:

//mobile and embedded /

```
$ sudo dpkg-reconfigure -p
low \
locales
```

The user interface is similar to the interface for setting the time zone.

Set up a time server. Many operations, including software package updates, rely on a correct date and time.

- To set the date and time automatically, first learn the identity of a network time server, and then run this command:

```
$ sudo ntpdate yourNTPServer
```

- To set the time on each boot, as superuser, edit `/etc/`

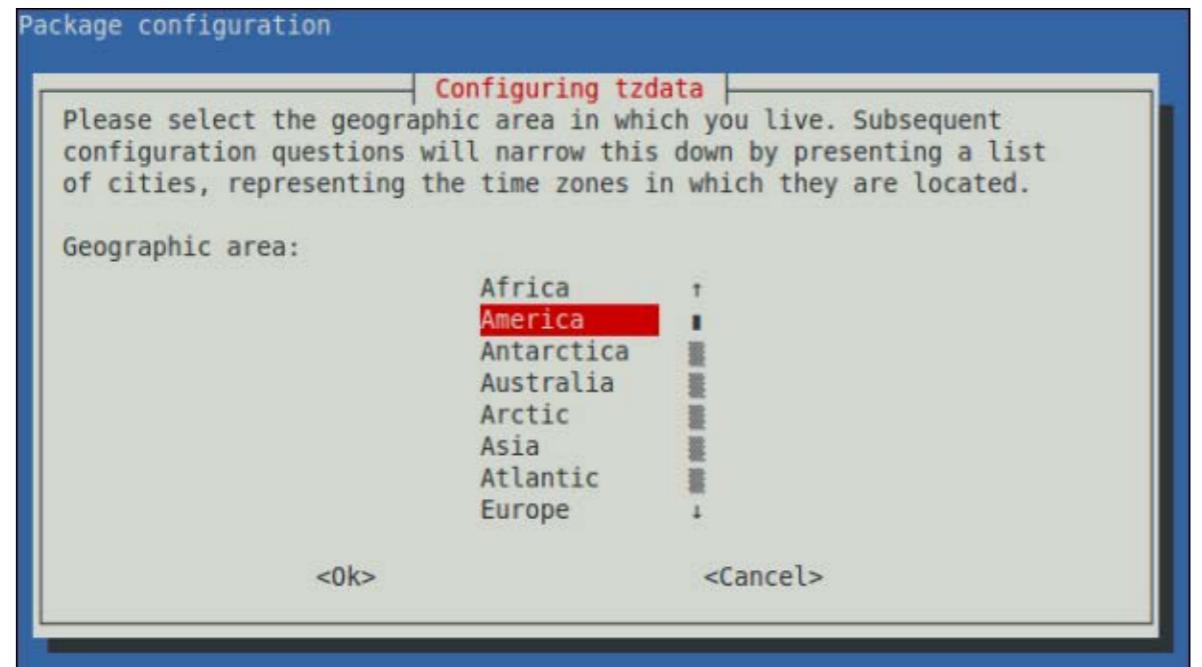


Figure 9

`rc.local` to add the following line before the script exits:

```
ntpdate yourNTPServer
```

Set up an internet proxy.

- Ping a well-known internet host, such as lego.com.
- If the host does not respond, find out whether your network is behind a proxy and, if necessary, set the environment variable `http_proxy` according to your network conventions.

Here is a hypothetical example:

```
$ sudo export \
http_proxy = \
'http://192.168.0.1:3128'
```

- To set the proxy each time you log in, add the `export` command to `~/.bash_profile` (or to the equivalent if you use a different shell).

Remap the keyboard.

- If keyboard keys are being misinterpreted or if you want to change the keyboard layout, run the following command:

```
$ sudo dpkg-reconfigure \
keyboard-configuration
```

- Make appropriate selections. The user interface is similar to the one described in the "Set the time zone and locale" section of this article.

Update the Debian packages.

- Before updating the packages distributed with Debian, as superuser, open `/etc/apt/sources.list` in a text editor. This file contains the repository locations of new and updated packages. As distributed, the locations have `ftp.uk` in their URLs. If you are not located in the United Kingdom, you can improve performance by changing the entries to a location that is physically closer to you. For example, you can globally change `ftp.uk` to `ftp.us` if you are in the United States. The

[Debian worldwide mirror sites page](#) has information about Debian repository locations.

- To update your installed packages, after verifying that the Raspberry Pi has an internet connection, enter these commands:

```
$ sudo apt-get update
$ sudo apt-get upgrade
```

- If you get a message that there are duplicates in the package list, open `/etc/apt/sources.list` in an editor again and comment out one of the duplicate entries. The entry containing more information is usually the best one to keep.

Automatically start the graphical user interface. If you want the GUI to start automatically when you log in locally to the Raspberry Pi, change the default runlevel as follows.

- As superuser, open `/etc/inittab` in a text editor.
- Find this entry:

```
# The default runlevel.
id:2:initdefault:
```

- Change the runlevel to 5:

```
# The default runlevel.
```

//mobile and embedded /

■ id:5:initdefault:

Note: Graphics consume a large chunk of the Raspberry Pi's RAM, on the order of 90 MB.

Add a different Web browser. If you experience difficulties with the default Midori browser, try Iceweasel:

■ \$ sudo apt-get update

■ \$ sudo apt-get install iceweasel

Enable sound. Raspberry Pi sound is not enabled by default. To enable it, do the following:

1. As superuser, open [/etc/modules](#) in a text editor.
2. Add this line to the end of the file:

■ snd_bcm2835

3. Save and close the file. Sound is enabled when you reboot the Raspberry Pi.

Conclusion

Being both affordable and compact, the Raspberry Pi is ideal for developing applications on an ARM-based device, and it has quickly risen in popularity since its initial release in early 2012. The cross-platform architecture of the Java SE for Embedded Devices runtime enables developers to create applications for the Raspberry

Pi just as easily as on other devices based on the ARM architecture or on one of its other supported architectures, including x86 and PowerPC. The Raspberry Pi's limited memory (256 MB RAM) and processing power (700 MHz CPU) make it a good development platform for embedded use cases and ideal for the Java SE for Embedded Devices runtime.

The advent of the Raspberry Pi device is an example of the increasing presence of ARM-based devices in both the embedded-device and desktop-device markets. To offer a solution for higher-end devices, the full-featured JDK provides support for the ARM platform architecture in the [Java SE 7 update 6 and later releases](#). </article>

LEARN MORE

- [Raspberry Pi](#)
- [Java SE for Embedded Devices](#)



TED NEWARD



Part 1

Building Actor-Based Systems Using the Akka Framework

Learn how to use Akka's open source actor model implementation to build distributed systems.

As just about any experienced Java developer will tell you, building a distributed system is hard. Building a distributed system that responds well to failure, scales well, handles all concurrency issues well, and still remains even remotely easy to understand comes pretty close to a working example of the word *impossible*.

Sun tried to build a model that made it easier to build a distributed, fault-tolerant, scalable, concurrent system, and they called it Enterprise JavaBeans (EJB). By most accounts, it didn't work well. Spring tried to create a simpler EJB approach, as did other "micro-containers." Despite their efforts, building a

distributed system that satisfies all of the above criteria still requires deep knowledge of the platform, language, and threading model.

Outside the Java world, however, interesting things were afoot. A "new" (to Java users) language and platform, Erlang, began gaining interest based on its successes in building distributed systems in the telecommunications industry. Telecoms aim for "five nines" availability, so any language or platform that can achieve such success is worth examining.

In this particular case, part of what came back was the *actor model*, which is a different way of approaching the construction of distributed sys-

tems. It is a new abstraction, one divorced from the traditional objects-in-threads model that underlies many of the other Java approaches to distributed systems. In essence, using actors, developers no longer think about objects calling methods on other objects across remote systems to obtain data and modify it. Instead, they focus on sending messages to actors operating independently in a share-nothing, immutable-state environment.

If that last sentence sounded a bit like gibberish, that's OK—it represents a different way of thinking about building systems, something we will explore in more detail. Specifically, we're going to investigate building actor-based systems using the Akka framework, which is part of the

Typesafe Stack that includes Scala, Akka, and the Play framework. If you're not familiar with Scala, check out [this Java Magazine article](#) for a succinct overview of the language or, for a longer, older take on Scala, check out my "[Busy Java Developer's Guide to Scala](#)" series.

Installing Akka and Scala

Akka is an open source actor model implementation, so the first step for using Akka is to make sure you have the [Akka library](#) downloaded and installed somewhere on your machine. If you're going to use Scala (as I will), you'll also need to install the [Scala bits](#).

For these examples, I'm using Akka 2.0.3 and Scala 2.9.2. If neither is currently on your machine, you can download the [Typesafe Stack](#) as a whole, which manages some of the dependencies

IN WITH THE NEW
All this flies directly in the face of traditional object orientation. **That's a good thing.**

//polyglot programmer /

and offers some additional features beyond just Akka.

Make sure that you have Scala available at the command prompt, and ensure that the Akka libraries are accessible to your [CLASSPATH](#). For simplicity's sake, I'll be explicit about which libraries we're compiling and running against, rather than letting sbt (Scala Build Tool) or Maven hide those details.

Hello, Akka (and Scala)

First, verify that Scala is installed by running `scala -version` at the command line, and make sure it reports at least version 2.9. Then, let's look at what is probably the simplest "Hello Akka" program that follows the style and ideas embodied in Akka (see **Listing 1**).

For now, don't worry about what the code does; let's make sure it builds and runs first. To build it, we'll need two libraries from the Akka distribution ([akka-actor-2.0.3.jar](#) and its sole dependency, [config-0.3.1.jar](#)). So for ease of use, copy them to the same directory where **Listing 1** (Main.scala) lives, and build. (If you're new to Scala, scalac, the command-line compiler, uses many of the same flags that javac uses. So pass in the JAR files using the `-classpath` option just as you would for javac.)

If it builds successfully, then run it (again, using scala `-classpath ...`

[Main](#), just as you would for Java), and lo and behold, a message to the console appears.

And . . . the program never returns. This is expected, and we'll fix it in a second, but to understand why requires first diving a bit into the concepts behind Akka.

Actors, to Your Places!

In a traditional object-based remoting system, we build objects that expose methods over some form of remoting protocol or API. Unfortunately, doing so requires that we think explicitly about a number of sticky issues.

For example, should the parameters be passed *by value* (that is, should what's passed in be a copy of the parameter objects and, if so, should it be a "deep" or "shallow" copy?) or *by reference* (that is, should what's passed in be a proxy to a remote object that lives elsewhere?). Also, can other remote clients call this same method on this same object during the time that the first client is calling this object's method? And . . .

In an actor-based system, by contrast, we never expose methods to the outside world. Instead, to simplify a bit, clients of an actor send messages to the actor, which are always "by value" copies, and they are processed in an entirely asynchronous fashion. By

LISTING 1

```
import akka.actor.{ ActorSystem, Actor, Props }

case object Start

object Main {
  def main(args: Array[String]): Unit = {
    val system = ActorSystem()
    system.actorOf(Props[HelloActor]) ! Start
  }
}

class HelloActor extends Actor {
  def receive = {
    case Start => System.out.println("Hello, Akka!")
  }
}
```

[Download all listings in this issue as text](#)

eliminating the shared mutable state that passing parameters by reference might introduce, and by eliminating direct access to object internals, we eliminate the potential for two clients to be accessing the same variables at the same time, which eliminates the need to worry about locking. By sticking with messages, instead of objects, as the parameters we pass around, we avoid accidentally handing a huge graph of domain objects across the wire.

To be clear, I use the term *object* to mean a traditional union of state plus behavior, such as with

a domain object, a model object, or a business object, as has been used in other systems, such as EJB or Servlets.

An *actor*, on the other hand, is simply an entity that responds to messages—it's never intended to be a domain object or to represent something in the business domain. It simply processes incoming messages. (Yes, an actor might in turn talk to a domain object, but that's getting ahead of ourselves.)

A *message* is essentially a data transfer object, or a parameter object, to use Martin Fowler's term. A message, contrary to tradi-



//polyglot programmer /

tional object-oriented principles, offers no encapsulation of its state, because it needs none—the whole point of the message is to convey data, not to hide it or offer any kind of processing.

From an implementation standpoint, yes, all three kinds of entities (objects, actors, and messages) are built using the same atoms (classes, fields, and methods) but in very different proportions. For example, if your actors are trying to track business data in fields, you're closer to building an object than an actor. If your messages are defining anything other than public fields (and maybe a few convenience methods such as `toString()` for easier diagnostic messages), you're closer again to building an object than a message.

And yes, all this flies directly in the face of traditional object orientation. That's a good thing.

Getting back to the example code, what we're doing here is creating an instance of an actor (`HelloActor`) and sending it a message (`Start`). This is the heart of actor-style programming: creating entities (actors) that respond to messages and, depending on the purpose of the message, replying back with values or passing those messages on to other actors.

Take careful note: At no point will one object directly invoke a

method on another—instead, message passing takes place, and any data inside those messages (of which there is none, yet) is copied instead of passed by reference.

In `main()`, we first initialize the Akka actors system by obtaining an instance of `ActorSystem`, and then we use `Props` (think of it as a dictionary of actors for now) to obtain an instance of the `HelloActor` actor.

Note that where we get this actor from is hidden from us—it could be local or remote. We don't care. (Obviously transmission times will be much longer if it's remote, but since actor-delivered messages are intended to be asynchronous, we shouldn't care).

Then, we use the `!` method on the actor to send the `Start` message to it, which tells it to print to the console.

`HelloActor` is the actor implementation, and it's pretty straightforward. As a class, it derives from the base class `Actor`, and defines a `receive` method that pattern-matches for incoming `Start` objects.

Akka can use any `Serializable` type as a message type, but the type-safe tendencies of Scala lead us to prefer using simple class types (`case class` and `case object` are perfect for this) rather than data-driven types that might be misinterpreted at runtime. For

LISTING 2

```
object Main {
  def main(args: Array[String]): Unit = {
    val system = ActorSystem()
    system.actorOf(Props[HelloActor]) ! "Start"
  }
}

class HelloActor extends Actor {
  def receive = {
    case "start" => System.out.println("Hello, Akka!")
  }
}
```

[Download all listings in this issue as text](#)

example, we could change the code to use `Strings` instead, as shown in Listing 2.

But the dangers here are exactly what we saw in the earlier code, because now that we're using a `String`—and the data held inside the `String`—to convey the message, typos and case-sensitivity become issues. Granted, we could set a “corporate standard” that says “all messages must be in uppercase,” but this becomes one more thing we have to remember, as opposed to something the compiler can check for us.

If we go back to the original code, trying to send a `Strat` object will yield a compiler error, whereas trying to send a `Strat` string will compile and just do nothing.

(Worse yet, even if you're unit testing the `HelloActor`, the tests won't catch it—the error is in the client code, not the actor.)

Cut! Horrible! Do It Again!

We have two problems with the code as it currently is. One is that it never terminates. The other is that it's a pretty limited example. What if we want to print more than just “Hello” or we want to repeat that message a few times, to name a few possible enhancements?

Fixing the first issue is easy: we have to tell the Akka runtime to stop. When the `HelloActor` was created, the Akka runtime spun up a collection of threads under the hood. Then, when a message was received by the Akka run-

//polyglot programmer /

time, it borrowed a thread from the pool, called the `HelloActor.receive()` method with the message as a parameter, and—when that returned—returned the thread back to the pool. This is normal behavior; no actor has an explicit thread of its own.

But the runtime was never told to stop, so let's fix that. Let's create a new message type, `Stop`, that tells the actor that it's time to go away now. Doing so requires only that we define a new message type, `Stop`, and that when the actor receives that message, it shuts down the Akka runtime, as shown in **Listing 3**.

Now, when the code is run, the actor prints to the console, and then the whole thing stops. Where did this “context” thing come from? We inherited it from [Actor](#)—every actor lives inside an Akka runtime context, and from that we can get a variety of interesting tidbits, such as a reference to the Akka system, which we then use to shut the Akka system down.

But let's also make the `HelloActor` slightly more powerful; let's break the messages up a little into an "initializing" sequence (`Start`), a "finished" sequence (`Stop`), and a "do something" sequence (`Message`) that carries with it some additional data. See **Listing 4.**

Note that in Listing 4, just to prove a point, I created the `RepeatMessage` to extend the `Message`, because this is common in some actor systems and was sometimes used in earlier Akka code examples. Scala 2.9 will issue a deprecation warning about this, however, and because the inheritance here isn't critical to what we're doing, if deprecation warnings bother you, you can remove the "extends" clause (and eliminate the "override," too). The rest of the code will work just as well.

However, there is one important aspect about messages in Akka that might go unnoticed: messages need to be immutable. There's no way to enforce this at compile time, unfortunately. But in Scala, case classes and objects are immutable by default (unless you explicitly make them mutable, which is a bad idea), making them perfect for message types.

Next, we need to modify the `HelloActor` to respond to these messages, as shown in **Listing 5**.

Again, it's all pattern matching, so if that part isn't clear, check out some of the available Scala references. The rest of the code—what we do after matching on a message type—should be pretty self-evident. Finally, there's the change to the client to start using it this way. (See **Listing 6.**)

LISTING 3 / **LISTING 4** / **LISTING 5** / **LISTING 6**

```
import akka.actor.{ ActorSystem, Actor, Props }

case object Start
case object Stop

object Main {
    def main(args: Array[String]): Unit = {
        val system = ActorSystem()
        val helloActor = system.actorOf(Props[HelloActor])
        helloActor ! Start
        helloActor ! Stop
    }
}

class HelloActor extends Actor {
    def receive = {
        case Start => System.out.println("Hello, Akka!")
        case Stop => context.system.shutdown()
    }
}
```



[Download all listings in this issue as text](#)

Again, we're using the `!` method in Akka (also called the "tell" style of message-send), which fires these messages off in a fire-and-forget fashion, because no reply is returned from any of them. This is the best approach to an actor-based system, because the lack of a reply means no blocking on the part of the client. Also, if the actor takes a bit longer to process one of them—imagine sending a `RepeatMessage("Really long message here", 100000)` to

the `HelloActor`—the client doesn't feel any of that performance hit, because it drops the message off into a mailbox inside of the Akka runtime and immediately returns.

Build it, run it, and bask in the glorious results.

Cut! Excellent! Next Scene!

Admittedly, calling these results “glorious” is a bit of a stretch—after all, all the demo is doing is just printing to the console, and how exciting is that, really? But the

//polyglot programmer /

concepts behind the demo highlight some of the basic, fundamental principles of actor-based programming: establishing actors, sending messages, avoiding mutable state, and avoiding shared objects or remote objects.

It's a different approach to programming than an expatriate Java EE programmer is used to, and more than anything, success with Akka means rewiring your brain to think in this new style and approach. If you want to understand the heart and soul of how an actor-based model works, this is all that you need to know.

Let's consider an example from the Akka distribution, which calculates the mathematical constant Pi (see **Listings 7a–7d**). We won't go over the math involved; we'll cover just the design.

Again, leaving alone the algorithm by which we calculate Pi, notice that the work is essentially broken down into a collection of **Worker** actors, who are given work by the **Master** actor, who spins off pieces of the work to each **Worker** actor and then collates the results and hands back the completed answer to the original client.

This is a common concurrency pattern, sometimes called the *Master-Slave pattern*. It's how the original Apache Web server worked, where a master lightweight pro-

cess would take incoming HTTP requests on port 80, hand the request and all of its necessary context to a slave process for processing, and then hand the results back over the port to the client. It's also how the SETI@Home screen saver works (a master hands out chunks of data over the internet to client machines signed up for the program, and each client analyzes the signal data and then hands back the results to the master, signaling the desire to process more data).

That's a Wrap! Print It!

There's a great deal more to the Akka runtime than what we saw here, but we explored the basic, fundamental principles of actor-based programming: establishing actors, sending messages, avoiding mutable state, and avoiding shared objects or remote objects.

In the next article, we'll look at how Akka provides more support than just the basic actor model shown above. We'll talk about supervision, Akka's Actor Models of how fault tolerance works and, of course, we'll explore a bit more Akka code just for fun. </article>

LEARN MORE

- [Akka](#)
- [Jonas Bonér on Akka](#)

LISTING 7a **LISTING 7b** **LISTING 7c** **LISTING 7d**

```
/**  
 * Copyright (C) 2009-2012 Typesafe Inc. <http://www.typesafe.com>  
 */  
package akka.tutorial.first.scala  
  
import akka.actor._  
import akka.routing.RoundRobinRouter  
import akka.util.Duration  
import akka.util.duration._  
  
object Pi extends App {  
  
    calculate(nrOfWorkers=4, nrOfElements=10000,  
              nrOfMessages=10000)  
  
    sealed trait PiMessage  
    case object Calculate extends PiMessage  
    case class Work(start: Int, nrOfElements: Int)  
        extends PiMessage  
    case class Result(value: Double) extends PiMessage  
    case class PiApproximation(pi: Double, duration: Duration)
```

[Download all listings in this issue as text](#)

//fix this /



In the September/October issue, Stephen Chin presented a [code challenge](#) around JavaFX Media APIs. He gave us a code snippet from the Application start method of the MediaQuiz class and asked us to figure out why the video clip does not play.



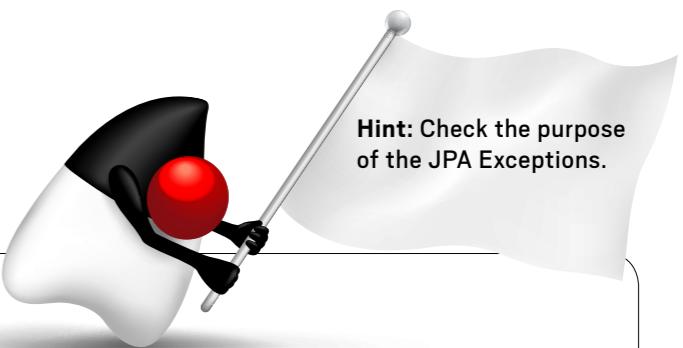
The correct answer is #3. While we have loaded the media and inserted the MediaView into the scene correctly, we have not actually started playing the media by calling play() on the MediaPlayer class.

This is an easy bug to overlook with three different media classes in use, but with a little detective work, it's not that hard to diagnose.

This issue's challenge comes from Jason Hunter (top left), author of *Java Servlet Programming*, 2nd Edition (O'Reilly Media) and deputy CTO at MarkLogic, and Boris Shukhat, applications programming manager, vice president, Bank of America Merrill Lynch.

1 THE PROBLEM

Porting JDBC code to use JPA requires a surprisingly different treatment of transactions rollbacks.



Hint: Check the purpose of the JPA Exceptions.

2 THE CODE

The program was originally designed to talk to a database via JDBC using the following fragment of code:

```
try {  
    statement.executeUpdate(sqlInsertStatement);  
    connection.commit(); log.info("Transaction committed");  
} catch(SQLException e) {connection.rollback();  
    log.info("Transaction rolled back");}
```

After several years the program was redesigned to use JPA:

```
entityTransaction.begin();  
try{  
    entityManager.createNativeQuery(sqlInsertStatement)  
        .executeUpdate();  
    entityTransaction.commit(); log.info("Transaction  
        committed");  
} catch(Exception e) {entityTransaction.rollback();  
    log.info("Transaction rolled back");}
```

As a result the code failed.

3 WHAT'S THE FIX?

- 1.) Check whether entityTransaction.isActive() is true before calling entityTransaction.rollback().
- 2.) Call entityTransaction.rollback() for RuntimeException.
- 3.) Do not call entityTransaction.rollback() for RollbackException.
- 4.) All of the above.

GOT THE ANSWER?

ART BY I-HUA CHEN

Look for the answer in the next issue. Or [submit](#) your own code challenge!

EDITORIAL**Editor in Chief**

Caroline Kvitka

Community EditorsCassandra Clark, Sonya Barry,
Yolande Poirier**Java in Action Editor**

Michelle Kovac

Technology Editors

Janice Heiss, Tori Wieldt

Contributing Writer

Kevin Farnham

Contributing Editors

Blair Campbell, Claire Breen, Karen Perkins

DESIGN**Senior Creative Director**

Francisco G Delgadillo

Senior Design Director

Suemi Lam

Design Director

Richard Merchán

Contributing Designers

Jaime Ferrand, Nicholas Pavkovic

Production Designers

Sheila Brennan, Kathy Cygnarowicz

ARTICLE SUBMISSIONIf you are interested in submitting an article, please [e-mail the editors](#).**SUBSCRIPTION INFORMATION**

Subscriptions are complimentary for qualified individuals who complete the subscription form.

MAGAZINE CUSTOMER SERVICEjava@halldata.com Phone +1.847.763.9635**PRIVACY**Oracle Publishing allows sharing of its mailing list with selected third parties. If you prefer that your mailing address or e-mail address not be included in this program, contact [Customer Service](#).

Copyright © 2012, Oracle and/or its affiliates. All Rights Reserved. No part of this publication may be reprinted or otherwise reproduced without permission from the editors. JAVA MAGAZINE IS PROVIDED ON AN "AS IS" BASIS. ORACLE EXPRESSLY DISCLAIMS ALL WARRANTIES, WHETHER EXPRESS OR IMPLIED. IN NO EVENT SHALL ORACLE BE LIABLE FOR ANY DAMAGES OF ANY KIND ARISING FROM YOUR USE OF OR RELIANCE ON ANY INFORMATION PROVIDED HEREIN. The information is intended to outline our general product direction. It is intended for information purposes only, and may not be incorporated into any contract. It is not a commitment to deliver any material, code, or functionality, and should not be relied upon in making purchasing decisions. The development, release, and timing of any features or functionality described for Oracle's products remains at the sole discretion of Oracle. Oracle and Java are registered trademarks of Oracle Corporation and/or its affiliates. Other names may be trademarks of their respective owners.

Java Magazine is published bimonthly with a free subscription price by Oracle, 500 Oracle Parkway, MS OPL-3C, Redwood City, CA 94065-1600.

Digital Publishing by Texterity

PUBLISHING**Vice President**

Jeff Spicer

Publisher

Jennifer Hamilton +1.650.506.3794

Audience Development and Operations Director

Karin Kinnear +1.650.506.1985

ADVERTISING SALES**Associate Publisher**

Kyle Walkenhorst +1.323.340.8585

Northwest and Central U.S.

Tom Cometa +1.510.339.2403

Southwest U.S. and LAD

Shaun Mehr +1.949.923.1660

Northeast U.S. and EMEA/APAC

Mark Makinney +1.805.709.4745

Advertising Sales Assistant

Cindy Elhaj +1.626.396.9400 x 201

Mailing-List Rentals

Contact your sales representative.

RESOURCES**Oracle Products**

+1.800.367.8674 (U.S./Canada)

Oracle Services

+1.888.283.0591 (U.S.)

Oracle Press Booksoraclepressbooks.com

COMMUNITY

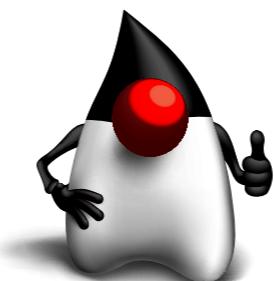
JAVA IN ACTION

ABOUT US



Learn Java On Demand

— Oracle University —



- ✓ Complete Classroom Content
- ✓ Highest Rated Instructors
- ✓ Immediate Online Access
- ✓ Search, Pause & Rewind Anytime

[Preview Now](#)